VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekcia ohraničujúcich kvádrov okolo objektov v obraze

Bounding Box Detection in Images

2020 Lukáš Mrvečka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Mrvečka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce obalových kvádrů kolem objektů v obraze
Bounding Box Detection in Images

Jazyk vypracování: slovenština

Zásady pro vypracování:

Cílem práce je vytvořit aplikaci pro hledání obalových kvádrů kolem zvoleného typu objektů (např. kvádr označující polohu auta na vozovce). Předpokládá se využití existujícího frameworku pro vytváření neuronových sítí (TensorFlow nebo Darknet) a knihovny OpenCV. Programovacím jazykem bude C/C++ nebo Python.

1. Nastudujte zvolený framework pro vytváření neuronových sítí a seznamte se s problematikou hledání obalových kvádrů.
2. Navrhněte svou vlastní nebo využijte již existující architekturu sítě umožňující získat odhad polohy obalové kvádru v obraze z jedné kamery.
3. Ověřte funkčnost aplikace na vhodném datasetu.
4. Postup a dosažené výsledky pečlivě popište v textové části práce.

Seznam doporučené odborné literatury:

[1] Li, Buyu, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. "GS3D: An Efficient 3D Object Detection Framework for Autonomous Driving." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1019-1028. 2019.
[2] Sochor, Jakub, Jakub Špaňhel, and Adam Herout. "Boxcars: Improving fine-grained recognition of vehicles using 3-d bounding boxes in traffic surveillance." IEEE Transactions on Intelligent Transportation Systems 20, no. 1 (2018): 97-108.
[3] Mousavian, Arsalan, Dragomir Anguelov, John Flynn, and Jana Kosecka. "3d bounding box estimation using deep learning and geometry." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7074-7082. 2017.
[4] Sochor, Jakub, Adam Herout, and Jiri Havel. "Boxcars: 3d boxes as cnn input for improved fine-grained vehicle recognition." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3006-3015. 2016.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 14. mája 2020 .....

Týmto by som chcel poďakovať vedúcemu diplomovej práce Ing. Tomášovi Fabiánovi, Ph.D za odbornú pomoc a konzultácie pri vytváraní tejto práce

Abstrakt

Hlavnou témou tejto práce je detekcia 3D ohraničujúcich kvádrov okolo objektov v obraze. Na začiatku sme spravili prieskum aby sme zistili, ktoré metódy detekcie sa v súčastnosti používajú a aké dosahujú výsledky. Pre našu prácu sme vybrali metódu DENSEBOX pretože je relatívne rýchla a výstup sa dá veľmi ľahko upraviť. O detekciu sa postará konvolučná neurónová sieť. Pri návrhu NN sme sa inšpirovali prácou [18] v ktorej NN detekuje objekty v rôznych mierkach. Použili sme framework TensorFlow, pre ktorý musela byť implementácia NN upravená, avšak architektúra ostala rovnaká. K vybranej metóde je dôležité vhodne zvoliť dataset. Rozhodli sme sa použiť KITTi dataset, ktorý je dostatočne obsiahly a dobre popísaný. Výsledkom je program, ktorý využíva neurónovú sieť na detekciu ohraničujúcich kvádrov rôznych veľkostí a zobrazenie výsledkov užívateľovi.

Kľúčové slová: DENSEBOX, KITTI dataset, detekcia objektov, spracovanie obrazu, Tensorflow, konvolučná sieť

Abstract

The main theme of this work is the detection of 3D bounding box around objects in image. At the beginning, we had to do a survey to find out which methods are currently being used and what results they are achieving. We chose the DENSEBOX method because it is relatively fast and the output is very easy to edit. The object detection is done by convolution neural network. During designing NN we were inspired by the work [18] in which NN detects objects at different scales. We used the Tensorflow framework, for which the NN implementation had to be modified but the architecture remained the same. It is important to select the dataset appropriately for the selected method. We decided to use a KITTI dataset that is sufficiently comprehensive and well described. The result of this work is program which uses neural network to detect bounding boxes various sizes and visualize the results.

Key Words: DENSEBOX, KITTI dataset, object detection, image processing, Tensorflow, CNN

Obsah

Zoznam použitých skratiek a symbolov

NN	–	Neural Network (neurónová sieť).
DNN	–	Deep Neural Network (neurónová sieť s viacerými skrytými vrstvami).
CNN	–	Convolution Neural Network (konvolučná neurónová sieť).
RPM	–	Region Proposal Method (metódy založené na hľadaní záujmových oblastí).
IoU	–	Intersection over Union (presnosť prekrytia dvoch 2D objektov).
PDE	–	Point Distance Evaluation (presnosť prekrytia dvoch 3D objektov založený na euklidovskej vzdialenosti).
AR	–	Augmented Reality (rozšírená realita).
RPN	–	Region Proposal Network (sieť, ktorá vyhľadáva záujmové oblasti).
RANSAC	–	RANdom SAmple Consensus (algoritmus určený na hľadanie najviac vyhovujúcich parametrov).
ML	–	Machine Learning (strojové učenie).

Použité značenie

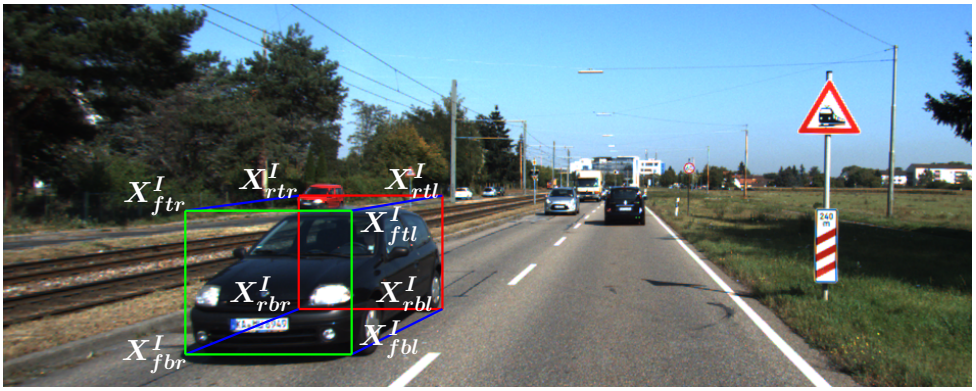
FOV	–	Field Of View (zorné pole neurónov vo výstupnej vrstve)
$\boldsymbol{X}$	–	vektor, matica
X	–	skalár
$\boldsymbol{x}$	–	vektor
x	–	skalár
()	–	vektor, matica
[]	–	súradnice
$\boldsymbol{X}^{I}_{ftl}$	–	bod v súradnicovom systéme obrázka (2D)
$\boldsymbol{X}^{W}_{ftl}$	–	bod v súradnicovom systéme auta (3D)

Zoznam obrázkov

Zoznam tabuliek

1 Úvod

Dopyt po autonómnych vozidlách rastie vo verejnej aj obchodnej oblasti. Ľudia stále viac požadujú bezpečnejšie a menej časovo náročné prostriedky a služby, ktoré by boli k dispozícii do niekoľkých minút. V obchodnej oblasti prichádza snaha o automatizáciu hlavne z dôvodu zvýšenia spoľahlivosti a maximalizáciu využitia prostriedkov. Napríklad autonómne nákladne vozidlá, ktoré by mohli jazdiť 24 hodín denne bez ľudského zásahu. Všetky dnešné vozidlá bez ohľadu na to či dokážu jazdiť samostatne, sú vybavené množstvom senzorov a kamier, ktoré poskytujú dostatok informácii. Spracovaním týchto informácii, či už samostatne alebo ich spojením [5] je možné získať doplňujúce informácie o objektoch (pozícia, veľkosť, trieda objektu), ktoré sa nachádzajú v zornom uhle použitých senzorov a kamier. Detekcia objektov patrí k základným schopnostiam autonómnych systémov, pretože sa musia naučiť vnímať okolité prostredie a na základe toho sa rozhodnúť a spraviť ďalší krok. Ďalším príkladom využitia takéhoto systému by mohla byť mestská infraštruktúra, kontrola dôležitých križovatiek a cestných úsekov. Informácie zo systému by sa dali využiť aj v prípade plánovania trasy cez GPS, prípadne pri plánovaní rozšírení cestných úsekov alebo ich zefektívnení. V uplynulých rokoch sa do popredia stále viac dostáva rozšírená realita (AR), ktorá využíva práve informácie o 3D objektoch, ich pozíciu a veľkosť. Témou tejto práce je detekcia automobilov a ich reprezentácia 3D ohraničením. 3D ohraničenie definujeme ako kváder, ktorý má 7 stupňov voľnosti (pozícia, veľkosť, rotácia okolo osi y). Takto označený objekt na vozovke môže byť chápaný ako prekážka pre autonómne vozidlo. Príklad takto označeného vozidla môžme vidieť na obr. 1



Obr. 1: Príklad ohraničujúceho kvádra s popisom vrcholov. Kváder rozlišuje prednú (zelenú) a zadnú (červenú) stranu auta

Na obrázku je zobrazený ohraničujúci kváder objektu, ktorý je výsledkom tejto práce. Zaoberáme sa detekciou vozidiel a preto môžme predpokladať, že všetky vozidlá a teda aj všetky ohraničujúce kvádre ležia na základnej rovine. Postup ako získať základnú rovinu a načo ju budeme potrebovať je bližšie popísaný v kapitole 3.3. Pre kontrolu, či je použitý postup správny máme dve možnosti. Jednou z nich je vizualizácia priamo do obrázka, v ktorom sa snažíme

nájsť objekty. Druhou možnosťou je porovnanie výsledkov po inverznej projekcii 3.7 s údajmi z datasetu. Prvá z možností je v našom prípade dostačujúca, keďže pracujeme s dátami z datasetu a v danom momente používame len jeden obrázok. Ďalším krokom je implementácia neurónovej siete, ktorá detekuje objekty v reálnom čase. Dokument je rozdelený na niekoľko častí. V úvode je výsledok prieskumu podobných riešení a ich porovnanie. Nasledujúce kapitoly sú zamerané na získanie potrebných údajov pre ďalšie spracovanie, ich reprezentácia a použitie. Nasledujú kapitoly, ktoré popisujú použitý framework, architektúru neurónovej siete a jej implementáciu. Kapitola 5 podrobne popisuje postup, ako z výstupu neurónovej siete získať potrebné údaje pre inverznú projekciu. V tejto kapitole bol predstavený aj nový spôsob ako zistiť presnosť siete založený na euklidovskej vzdialenosti bodov. Počas implementácie a testovania sme si všimli niekoľko zaujímavostí, ktoré sú bližšie popísané v kapitole 6.

2 Obdobné riešenia

Neurónové siete (NN) nie sú žiadnou novinkou v dnešnej dobe. Ich história siaha až do roku 1943 kedy Warren McCulloch a Walter Pitts predstavili prvý model neurónovej siete. V priebehu dekád sa neurónové siete vďaka svojmu širokému využitiu dostávajú do všetkých oblastí priemyslu. Spracovanie obrazu nie je výnimkou. Prvé konvolučné siete, na klasifikáciu čísel boli predstavené už v roku 1989 [15]. Ďalší veľký moment prišiel v 2012 kedy Krizhevsky predstavil svoj model konvolučnej neurónovej siete [14]. Model sa skladal s 5 konvolučných vrstiev a dvoch „fully-connected“ vrstiev. Navrhnutý model dosiahol asi o 10% lepšiu úspešnosť v klasifikácii ako vtedajšie riešenia založené na metóde SIFT [22]. Tento úspech by však nebol možný bez dostatočne veľkého datasetu a výpočtového výkonu, ktoré prinášajú prevažne grafické karty. Od prelomu v roku 2012 bolo navrhnutých mnoho NN, ktoré boli schopné klasifikovať obrázky do rôznych skupín iba na základe prechodu obrázka cez konvolučné vrstvy, bez nutnosti zložitého pred-spracovania vstupu. Navrhnuté riešenia sú dostačujúce ak chceme klasifikovať obrázok ako jeden objekt. Na obrázku sa však môže nachádzať viacero objektov, rôznej veľkosti, či dokonca rôznej skupiny. Môžu nastať situácie, keď objekt bude obsahovať iný objekt. V prípade, že obrázky obsahujú viacero objektov, ktoré chceme klasifikovať je nutné ich v prvom kroku nájsť. Existujúce metódy, ktoré slúžia na klasifikáciu prípadne detekciu jednotlivých objektov môžme rozdeliť na dve skupiny. Hľadanie záujmových oblastí (RPM - Region Proposal Method's) alebo tzv „end-to-end“ metódy, ktoré na základe vlastností obrázka dokážu tieto objekty nájsť a klasifikovať.

2.1 Metódy RPM

Krizhevsky prácou [14] ukázal, že NN sú silným nástrojom pri klasifikácii objektov. Ako už bolo spomenuté v kapitole 2, aby mohol byť objekt klasifikovaný musí byť najskôr nájdený v obraze. Hľadanie objektov a záujmových oblastí pomocou metód RPM je založené na posúvaní pomyselného okna po obrázku a extrahovanie jednotlivých častí, ktoré sa dajú jednoducho klasifikovať. Tzv. „sliding-window“. Touto metódou je nutné prejsť celý obraz, aby sa našli všetky možné objekty. Obraz je nutné prejsť niekoľkokrát s rôznou veľkosťou okna, aby boli detegované objekty rôznych veľkostí. Počet takto extrahovaných častí narastá s komplexnosťou scény. V niektorých prípadoch môže počet takto extrahovaných častí dosiahnúť až 100 000 [26]. Pri metódach RPM sa kladie dôraz hlavne na zníženie počtu záujmových oblastí, a čo najmenšie ohraničenie objektu. Hľadanie takýchto oblastí je veľmi výpočtovo náročné, navyše na každú nájdenú oblasť je nutné v druhej fáze použiť nejaký typ klasifikátora, ktorý objekt v tejto časti identifikuje. Boli preto navrhnuté rôzne spôsoby ako hľadanie RPM oblastí zrýchliť a zredukovať ich počet. Každá záujmová oblasť disponuje špecifickým indexom „objectnes“ [26], ktorý je možné chápať ako pravdepodobnosť, že v danej časti obrazu sa nachádza nejaký objekt. Index je možné využiť na odstránenie oblastí s nízkou hodnotou tohto indexu a tak znížiť počet záujmových oblastí. Jedným z takých spôsobov je aj „selective search“ [26], ktorý využíva segmentáciu obrazu na

hľadanie RPM oblastí.

2.1.1 R-CNN

Metóda R-CNN [11] je rozdelená na tri časti. Prvú časť tvorí práve spomínaný „selective search“, ktorého výstupom je v tomto prípade okolo 2 000 RPM oblastí pre každý obrázok. V druhej časti je z každej RPM oblasti vytvorený vektor s veľkosťou 4 096 s využitím frameworku CAFFE a konvolučnej siete podľa [14]. Na záver boli použité SVN klasifikátory. Presnosť tejto metódy bola celkom dobrá avšak čas 18 s potrebný na spracovanie jedného obrázka je nedostačujúci.
Konvolučná sieť v prípade R-CNN vyžaduje fixnú veľkosť obrázka 224×224. Autori [12] prišli s návrhom siete SppNET, ktorá nevyžaduje fixnú veľkosť vstupu. Metóda SppNet generuje vektor fixnej veľkosti, ktorý reprezentuje daný obrázok. Vstup môže mať ľubovoľnú veľkosť. Konvolúcia sa aplikuje iba raz na vstupný obraz (nie na každú časť ako v prípade R-CNN), z ktorého sa následne generuje vektor, ktorý ho reprezentuje. Metóda SppNET dosahuje $24 - 102$ krát rýchlejšie výsledky pri porovnateľnej presnosti.
Tento fakt inšpiroval autorov R-CNN, ktorí nahradili prvú časť svojho riešenia („selective search“) SppNET a vytvorili tak Fast R-CNN [10]. Vďaka využitiu SppNET bola metóda Fast R-CNN oveľa rýchlejšia. Klasifikátor v tomto prípade bol nahradený „fully-connected“ vrstvou, ktorá určovala pravdepodobnosť nejakej skupiny objektov a súradnice ohraničenia objektu.
V priebehu vývoja a zdokonaľovania R-CNN prišiel Karen Simonyan s návrhom plne konvolučnej siete [24] s využitím konvolúcie 3×3 a vytvoril tak VGGNet. Navrhnutá sieť mala 19 vrstiev a dosahovala lepšie výsledky v porovnaní s R-CNN.
Na základe VGGNet bola vytvorená metóda Faster R-CNN [20]. „Selective search“ bol v tomto prípade nahradený sieťou RPN, ktorá vyhľadávala záujmové oblasti. RPN je v podstate metóda „sliding window“, ktorej výstupom je index „objectnes“ a ohraničenie objektu definované veľkosťou a pomerom strán. Výsledok RPN je následne klasifikovaný pomocou Fast R-CNN. Spojenie týchto algoritmov dosahuje rýchlosť asi 15 snímkov za sekundu a úspešnosť $80 - 90\,\%$.

2.1.2 Metódy detekcie 3D ohraničujúcich kvádrov

Metóda 3DOP [4], ktorá bola predstavená v roku 2015 využíva stereo kamery na vytvorenie 3D „point cloud“. Vytvorený „point cloud“ sa používa na výpočet rovnice roviny (všetky hľadané objekty autá, chodci majú spodnú stenu ohraničujúceho kvádra na tejto rovine) a ohodnotenie objektov, ktoré sa nachádzajú v mieste kde „point cloud“ dosahuje najvyššiu hustotu. Metóda 3DOP využíva upravenú architektúru VGG-16 na ohodnotenie ohraničujúcich kvádrov a predikovanie orientácie objektov. Presnosť 3DOP sa pohybuje v intervale (80 %, 90 %) no jej časová náročnosť neumožňuje využitie v real-time aplikáciach podľa tabuľky 1.

Mono3D je adaptácia 3DOP [3]. Stereo obraz nie je viac nutnosťou, avšak predpokladá sa zna-

losť rovnice základnej roviny. Taktiež odpadá nutnosť vytvárať „point cloud“. Na vstupe bola aplikovaná segmentácia obrazu pomocou plne konvolučnej siete SegNet [2]. Pozície objektov boli odvodené dvoma spôsobmi, z pohľadu kamery a z vtáčej perspektívy, čo umožnilo identifikovať aj objekty, ktoré boli veľmi slabo viditeľné. Pre každú triedu objektov bol použitý samostatný SVN klasifikátor. Mono3D, adaptácia 3DOP, dosiahla o niečo lepšie výsledky, no za cenu ešte vyššieho času detekcie podľa tabuľky 1.

Metódu Deep3DBox [17] je tiež možné zaradiť do skupiny RPM. Na vstupe nie je žiaden algoritmus na výber záujmových oblastí ako v prípade predošlých metód. Vstup tvoria 2D ohraničenia objektov. Metóda Deep3DBOX využíva predpoklad, že akékoľvek 3D ohraničenie musí byt maximálne také veľké ako je 2D ohraničenie na danom objekte. Používa DNN na odhad orientácie a veľkosti 3D ohraničenia z 2D ohraničenia. V 2D ohraničení je zakódovaná veľkosť objektu a orientácia voči kamere, čo uľahčuje projekciu 3D ohraničujúceho kvádra. Deep3DBox dosahuje podobné výsledky ako ostatné metódy avšak za cenu vyššieho času potrebného na detekciu podľa tabuľky 1.

2.2 Metódy end-to-end

Pri metódach „end-to-end“ ide všeobecne o snahu odstrániť nutnosť použitia RPM algoritmu na hľadanie záujmových oblastí. Algoritmy RPM sa v týchto prípadoch nahrádzajú prevažne konvolučnými sieťami, ktorých výstup sa delí na niekoľko častí podľa veľkosti hľadaných objektov.

Metóda F-ConvNet využíva na detekciu 3D ohraničení algoritmus posúvania zrezaných ihlanov pozdĺž osi z. Na vstupe sa predpokladá znalosť 2D ohraničení z ktorých sú extrahované skupiny bodov pomocou zrezaných ihlanov. Z každej takto vytvorenej skupiny bodov je následne pomocou PointNet [19] vytvorený vektor vlastností špecifický pre každú skupinu bodov. V tretej časti návrhu sa nachádza konvolučná sieť, ktorá berie ako vstup 2D vektor. Aby bolo možné využiť takto navrhnutú konvolučnú sieť je nutné upraviť výstup PointNet a vytvoriť z 1D vektora 2D. Na výtupe konvolučnej siete je trieda objektu a pozícia 3D ohraničenia. F-ConvNet dosahuje porovnateľné výsledky s ostatnými metódami za kratší čas podľa tabuľky 1.

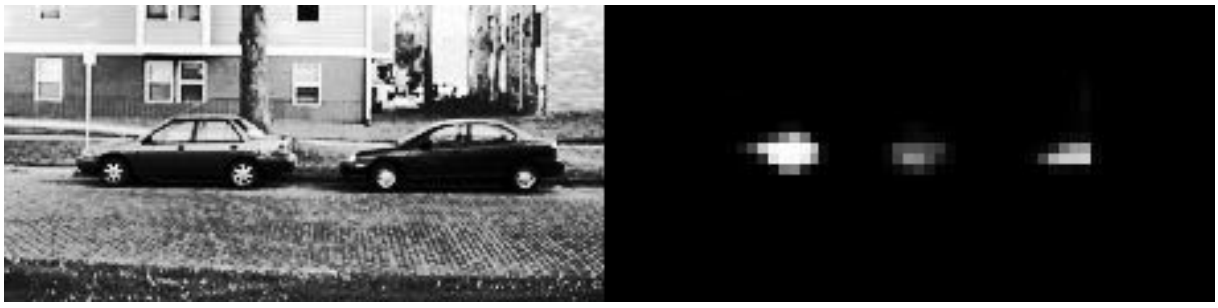
Metóda PointRCNN [23] je rozdelená na dve časti. V prvej fáze využíva „point cloud“ ako vstup algoritmu, z ktorého generuje masku 3D ohraničujúcich kvádrov. S pomocou vytvorenej masky sú vygenerované 3D ohraničenia, ktoré slúžia ako „ground-truth“. Navrhnutý postup odstraňuje nutnosť veľkého množstva 3D ohraničení z datasetu. V druhej fáze algoritmu sa jednotlivé body 3D ohraničení z prvej fázy združujú a prevádzajú na kanonické súradnice. Pri tomto procese sa súradnice kombinujú so segmentačnou maskou. Kombinácia so segmentačnou maskou zabezpečí relatívne zdokonalenie odhadovaných súradníc vzhľadom k pozícii jednotlivých bodov. Metóda PointRCNN sa ukázala ako veľmi rýchla a presná podľa tabuľky 1.

Metóda	Úspešnosť (%)	TTD (s)
3DOP	92,96	3,0
Mono3D	94,52	4,2
F-ConvNet	95,85	0,47
PointRCNN	96,70	0,1
Deep3DBox	94,71	1,5

Tabuľka 1: Obdobné riešenia, ich úspešnosť a čas detekcie jedného obrázka [8]

2.3 DenseBox

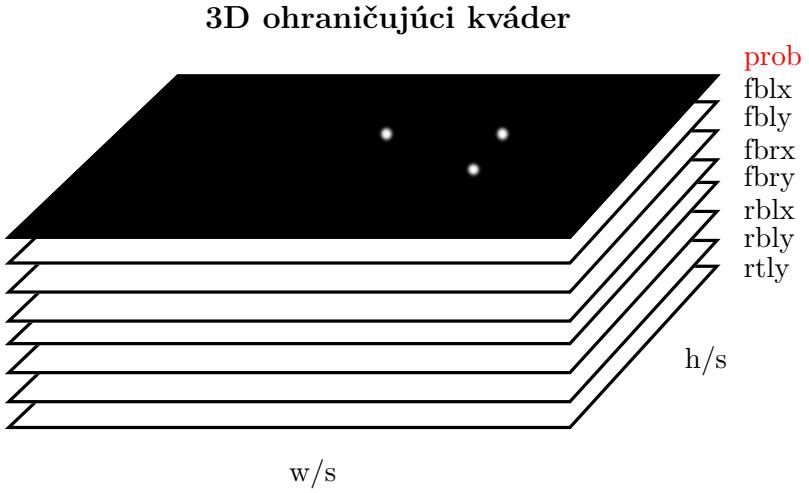
Detekcia automobilov pre autonómne vozidlá má určité požiadavky, ktoré musia byť splnené, aby bol detektor použiteľný. Musí byť schopný rozpoznať autá vo všetkých možných prípadoch, musí si poradiť aj v prípadoch, keď vozidlo môže byť z časti prekryté nejakou prekážkou. Najväčším problémom je nasvietenie scény. Problém môže nastať, ak by svetlo mierilo priamo do objektívu alebo v noci na slabo osvetlenej ceste. Detektor musí byť dostatočne rýchly robustný aby vedel včas poskytnúť informácie o ceste. V súčastnosti jedným z najvýkonnejších publikovaných systémov na detekciu je DenseBox [13]. Ako výstup používa pravdepodobnostnú mapu centier objektov naprieč celým obrazom 2. Detegované objekty je možné potom reprezentovať ako maximá z máp pravdepodobnosti. Každý pixel v pravdepodobnostnej mape predstavuje potencionálny objekt. Pomocou algoritmu „non-maxima-suppresion“ je možné počet potencionálnych objektov zredukovať a získať tak tie s najlepšou pravdepodobnosťou. Keďže sa jedná o plne konvolučnú sieť tak tento postup je možné aplikovať na obraz ľubovoľnej veľkosti. Vzhľadom k tomu, že výstup z DenseBox môžme ľahko použiť na odhad polohy objektov, rozhodli sme ho použiť aj v tejto práci.



Obr. 2: Vstupný obrázok(vľavo), príklad pravdepodobnostnej mapy DenseBox(vpravo). Nenulové pixely sa zhlukujú v okolí stredu objektu, kde každý pixel predstavuje objekt s nejakou pravdepodobnosťou. Zdroj [18]

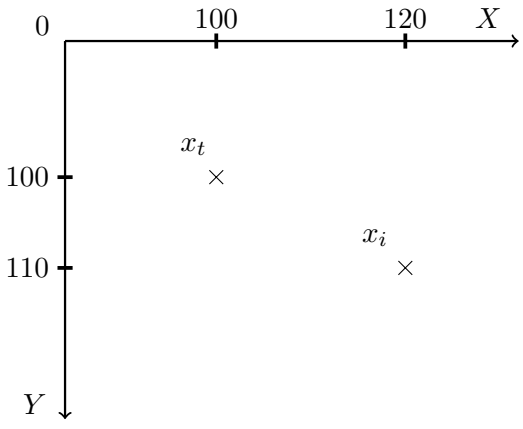
Metóda DenseBox je relatívne jednoduchá na implementáciu a jej výstup je možné upraviť podľa aktuálne riešeného problému. Základná implementácia DenseBox je prispôsobená na detekciu 2D objektov, keďže v práci sa zaoberáme detekciou 3D objektov musíme implementáciu

trocha upraviť. Ako referenčnú implementáciu sme zvolili [18]. V tomto prípade bola metóda detekcie upravená tak aby dokázala detegovať objekty rôznych veľkostí v jednom prechode. Detekcia objektov rôznych veľkostí je zabezpečená pomocou mierky, ktorá je definovaná na začiatku. Podľa toho aké objekty chceme detegovať môže byť mierka $2, 4, 8, 16$. Čím je mierka väčšia tým väčšie objekty deteguje. Výstup takto upravenej metódy má na výstupe 8 pravdepodobnostných máp. Zobecnený príklad výstupu je zobrazený na obr. 3. Prvá vrstva definuje pravdepodobnosť (spokojnosť) metódy s nájdeným objektom. 2. až 8. vrstva nesú informácie o x a y súradniciach jednotlivých bodov.



Obr. 3: Štruktúra výstupu upravenej DenseBox metódy na detekciu 3D ohraničujúcich kvádrov [18]

Výstup obsahuje 8 pravdepodobnostných máp, čo odpovedá trom vrcholom na základnej rovine a y súradnici jedného bodu z hornej steny kvádra, tak ako je to popísané v kapitole 3.5. Prvá *prob* vrstva určuje spokojnosť metódy s nájdeným objektom. Ďalšie vrstvy určujú rozdiel medzi súradnicami nájdeného bodu z DNN x_i a originálneho bodu z datasetu x_t 4.



Obr. 4: Výstup upravenej DenseBox metódy na detekciu 3D ohraničujúcich kvádrov

Predpokladajme x súradnicu nájdeného bodu fbl 120 a x súradnicu originálneho bodu 100, potom hodnota v odpovedajúcej vrstve $fblx$ bude $120 - 100 = 20$. Táto x-ová súradnica nájdeného bodu fbl je podľa toho posunutá o 20 px doprava. Túto hodnotu je možné chápať ako chybu DNN.

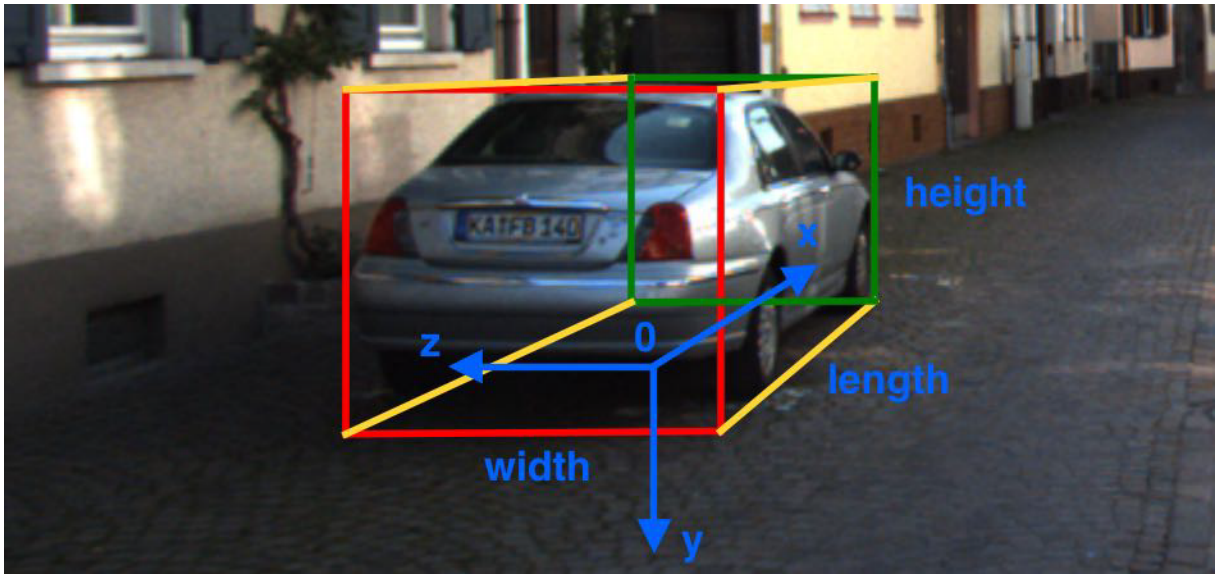
Pre popis vrcholov je použitá konvencia podľa tabuľky 2

Počet znakov	Hodnota	Popis
1	f alebo r	Predná (f) alebo zadná (r) stena kvádra
1	t alebo b	Horná (t) alebo spodná (b) stena kvádra
1	l alebo r	Ľavý (l) alebo pravý (r) bod v smere jazdy

Tabuľka 2: Použitá konvencia označovania vrcholov

3 KITTI Dataset

KITTI dataset [7] bol predstavený v roku 2012 inštitútom Karlsruhe. Pozostáva z obrázkov extrahovaných z video sekvencii zaznamenaných v uliciach Karlsruhe. Obsahuje veľmi presné anotácie, ktoré poskytujú dôležité informácie o objektoch na obrázku (polohu, veľkosť, v prípade ohraničujúcich kvádrov aj rotáciu vzhľadom ku kamere). Dataset je rozdelený na trénovacie a testovacie obrázky. Anotácie sú dostupné iba pre trénovaciu časť, pretože v súčastnosti stále prebieha súťaž na testovacej časti datasetu. Obrázky majú veľmi vysoký pomer šírka/výška: 1240×375, čo robí tento dataset jedinečný. Obrazové premietacie matice (Image projection matrices) sú k dispozícii pre trénovaciu aj testovaciu časť datasetu, vďaka tomu môžme spraviť rekonštrukciu z obrázka do 3D. Anotácie obsahujú informácie o 2D a 3D ohraničujúcich rámcov pre niekoľko tried objektov, z ktorých boli vybrané len informácie pre autá, ktoré budú použité v tejto práci. V prípade tejto práce stačia informácie pre 3D ohraničujúce kvádre (3D pozícia stredu spodnej steny kvádra, rotácia okolo osi y, šírka, výška a dĺžka). Údaje o pozícii a veľkosti objektu sú uvedené v metroch.



Obr. 5: Základné parametre pre 3D ohraničujúce kvádre. Anotácie KITTI datasetu obsahujú informácie o pozícii stredu spodnej steny kvádra **0** v 3D, výšku, šírku a dĺžku objektu. Z týchto informácii je možné získať pozície bodov v lokálnom súradnicovom systéme podľa kapitoly 3.6 a premietnuť ich do obrázka. KITTI dataset používa ľavo-ruký (left-handed) súradnicový systém otočený o 90 stupňov, to znamená, že vzdialenosť objektu od kamery určuje súradnica x a nie z. Zdroj [18]

Podľa obr. 5 a anotácii bol vytvorený ohraničujúci kváder v súradnicovom systéme auta a pomocou obrazovej projekčnej matice, rotačnej matice a translačného vektora je možné previesť kváder do súradnicového systému obrázka. K tomu bola použitá rovnica:

$$\boldsymbol{X}^W_{world} = \begin{pmatrix} \boldsymbol{R}_y & \boldsymbol{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{X}^W_{car} \\ 1 \end{pmatrix},$$

kde $\boldsymbol{X}^W_{world}$ sú vrcholy kvádra v súradnicovom systéme auta posunuté a otočené podľa predpisu na predpokladanú pozíciu, $\boldsymbol{R}_y$ predstavuje rotačnú maticu okolo osi y, $\boldsymbol{t}$ je translačný vektor, ktorý zabezpečí posun kvádra, $\boldsymbol{X}^W_{car}$ je matica vrcholov ohraničujúceho kvádra, kde každý stĺpec predstavuje jeden vrchol kvádra v súradnicovom systéme auta. Tieto vrcholy sa potom pomocou obrazovej projekčnej matice $\boldsymbol{P}$, ktorá je súčasťou anotácii KITTI dataset-u premietnu do obrázka pomocou rovnice

$$\begin{aligned} \boldsymbol{X}_h &= \boldsymbol{P} \cdot \boldsymbol{X}^W_{world} \\ \boldsymbol{X}^I_{image} &= \frac{\boldsymbol{X}_h}{\boldsymbol{X}_h[2]}. \end{aligned}$$

Projekčná matica $\boldsymbol{P}$ sa používa na prevod medzi 3D súradnicami a súradnicami obrázka ako je to popísané vyššie. Projekčná matica sa využíva aj v opačnom smere pri prevode zo súradníc obrázka do 3D, ako je to popísané v kapitole 3.7. Projekčná matica je jedinečná pre každý obrázok. Ako príklad je uvedená projekčná matica pre obrázok *000000.png*:

$$\begin{pmatrix} 707{,}0493 & 0{,}0000 & 604{,}0814 & 45{,}7583 \\ 0{,}0000 & 707{,}0493 & 180{,}5066 & -0{,}3454 \\ 0{,}0000 & 0{,}0000 & 1{,}0000 & 0{,}004981 \end{pmatrix}.$$

3.1 Popis anotácii dataset-u (label)

Anotačný súbor obsahuje informácie o všetkých objektoch na obrázku. Informácie pre daný objekt sú vždy na samostatnom riadku oddelené medzerami. Keďže sa zaoberáme detekciou vozidiel z datasetu boli vybrané len informácie o vozidlách. Anotačný súbor obsahuje nasledujúce informácie:

Car 0.00 0 1.84 303.95 181.88 465.56 292.30 1.52 1.55 3.79 $-$ 3.59 1.69 12.01 1.56

Počet hodnôt	Popis	Hodnota
1	Type (typ) - objektu	Car, Van, Truck, Pedestrian, Misc, DontCare
1	truncated (neúplnosť objektu)	0 (celý objekt je na obrázku) -> 1 (objektu je mimo obrázka, mimo zorného uhla kamery)
1	occluded (viditeľnosť)	0 (úplne viditeľný) - 3 (neznámy objekt)
1	alpha (pozorovací uhol)	pozorovací uhol vzhľadom ku kamere
4	bbox (2D ohraničujúci kváder)	$x,\ y$ ľavého horného a pravého dolného vrcholu kvádra v pixeloch
3	dimension (veľkosť)	veľkosť objektu v metroch
3	location (poloha)	stred spodnej steny kvádra v metroch $[x,\ y,\ z]$
1	rotation_y (rotácia objektu)	rotácia objektu okolo osy y

Tabuľka 3: Štruktúra anotácii KITTI datasetu

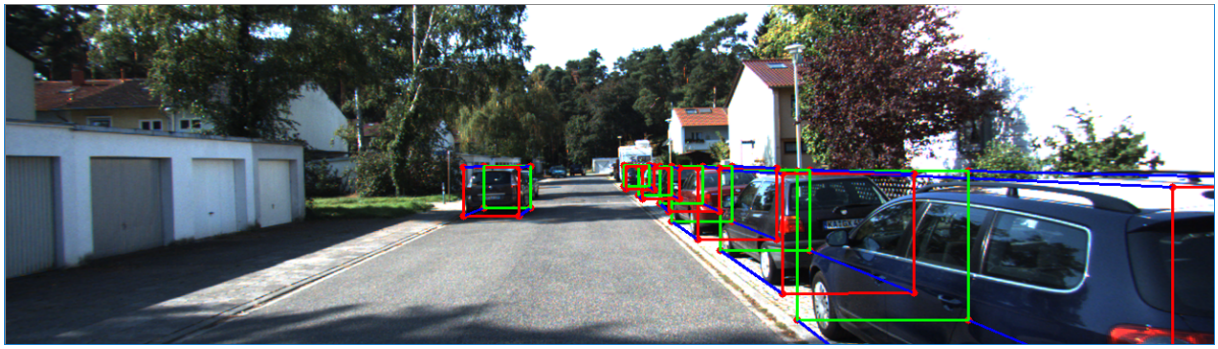
3.2 Reprezentácia dát

V tejto práci bol použitý dataset KITTI, pretože obsahuje dostatok údajov na trénovanie a testovanie navrhnutej siete. Tvorba datasetu a príprava trénovacích dát sa ukázala ako kľúčový faktor pri úspešnom trénovaní siete. Umelé zväčšenie datasetu využitím efektu rozmazania prípadne zvýraznenie niektorej farebnej zložky RGB môže viesť k robustnejšiemu modelu teda vyššej miere všeobecnosti trénovania. Vďaka rozsiahlosti datasetu a variabilite obrázkov úprava datasetu nebola nutná. Krátky popis datasetu sa nachádza v kapitole 3. Popis obrázkov v datasete obsahuje množstvo informácii, ktoré sú pri trénovani redundantné, preto je nutné ich odstrániť. Odstránenie nadbytočných údajov odľahčí nielen samotnú sieť ale aj zníži pamäťovú náročnosť programu.
Práca je zameraná na detekciu automobilov, keďže dataset obsahuje aj množstvo iných objektov, je nutné ich odstrániť. Pri pred-spracovaní datasetu boli odstránené objekty označené ako „Don't know“ teda neznámy objekt. V prípade týchto objektov môže ísť o rôzne prekážky na ceste alebo autá, ktoré sú prekryté iným objektom a je ťažké ich identifikovať. Objekty, ktoré sa podľa anotácii datasetu prekrývajú na viac ako 75 % boli tiež odstránené. Zvyšné objekty boli prevedené do nového formátu BB3TXT podľa kapitoly 3.4.

3.2.1 3D ohraničujúci kváder

3D ohraničujúce kvádre sú najmenšie pravouhlé kvádre, ktoré ohraničujú nejaké objekty. Objekty, ktoré sa prekrývajú na menej ako 75 % nie je možné ignorovať a musia byť zahrnuté vo výsledku. Ohraničujúci kváder má rozdelenú prednú a zadnú stranu auta aby bolo možné určiť

smer auta. Táto informácia je dôležitá aj pri zisťovaní presnosti siete v kapitole 5.1. Je veľmi dôležité aby ohraničujúci kváder bol položený na základnej rovine z dôvodu projekcie vrcholov (kapitola 3.7) z výstupu siete. Keďže je práca zameraná na detekciu áut a v súčastnosti neexistujú žiadne lietajúce autá, môžme pre jednoduchosť predpokladať že všetky autá sa nachádzajú na jednej (spoločnej) základnej rovine.



Obr. 6: Príklad z KITTI datasetu s 3D ohraničujúcimi kvádrami. Zelená - predná stena, červená - zadná stena kvádra

3.3 Výpočet rovnice roviny - RANSAC algoritmus

Použitá reprezentácia 3D ohraničenia vyžaduje rovnicu roviny aby bolo možné rekonštruovať ohraničujúce boxy v 3D svete. Rovnicu roviny je možné si predstaviť ako rovinu, na ktorej ležia všetky objekty, ktorých polohu sa snažíme odhadnúť. Pre každý obraz je nutné poznať rovnicu základnej roviny. Táto informácia nie je uvedená v KITTI dataset-e a preto je dôležité ju vypočítať [16]. Práca predpokladá, že počas celého vytvárania KITTI dataset-u bola použitá rovnaká kamera, výška umiestnenia kamery a sklon voči vozovke. Na základe tohto predpokladu je možné povedať, že rovnica roviny bude rovnaká pre celý dataset. Takto definovanú rovinu je možné použiť pri inverznej projekcii bodov z obrázka späť do 3D sveta, čo môžme použiť na kontrolu vypočítaných bodov. Každý ohraničujúci kváder obsahuje 4 vrcholy, ktoré ležia na tejto rovine, celkovo je k dispozícii 158 388 vrcholov. Na získanie koeficientov základnej roviny bol použitý RANSAC algoritmus. Na vstupe je zadefinovaný počet iterácii N a množina vrcholov $\boldsymbol{G}$ (vrcholy, ktoré ležia na základnej rovine). Rovina je definovaná pomocou troch vrcholov $\boldsymbol{p_1}$, $\boldsymbol{p_2}$, $\boldsymbol{p_3}$ z množiny $\boldsymbol{G}$ a je možné ju vypočítať ako:

$$\begin{aligned} \boldsymbol{l}_1 &= \boldsymbol{p}_2 - \boldsymbol{p}_1 \\ \boldsymbol{l}_2 &= \boldsymbol{p}_3 - \boldsymbol{p}_1 \\ \boldsymbol{n} &= \boldsymbol{l}_1 \times \boldsymbol{l}_2 \\ d &= -(\boldsymbol{n} \cdot \boldsymbol{p}_1) \end{aligned}$$

Príklad algoritmu RANSAC:

- Vstup: N, $G = [x_1,\ y_1,\ z_1]^T$, ..., $[x_m,\ y_m,\ z_m]^T$
- Inicializácia: $dst_{min} = \infty$
- For $i = 1$, ..., N
  1. Náhodne vybrané 3 vrcholy z množiny $\boldsymbol{G}$
  2. Vypočítame koeficienty rovnice $ax + by + cz + d = 0$, kde x, y a z sú definované vybranými vrcholmi
  3. Spočítané vzdialenosti dst všetkých vrcholov z $\boldsymbol{G}$ od kandidátskej roviny
  4. Ak $dst < dst_{min}$ tak $dst_{min} = dst$ a koeficienty a, b, c, d sa uložia ako najviac vyhovujúce
- Na výstupe sú najviac vyhovujúce koeficienty základnej roviny

Z výstupu algoritmu je možné zistiť normálu roviny ako $\boldsymbol{n} = (a,\ b,\ c)^T$. Sumu kvadrátov vzdialeností všetkých vrcholov od predpokladanej roviny, ktorá má byť minimálna je možné zistiť podľa rovnice:

$$dst = \sum_{i=1}^{M}((a,\ b,\ c,\ d)(x_i,\ y_i,\ z_i,\ 1)^T)^2$$

3.4 Prevod údajov z KITTI datasetu na BB3TXT formát

Údaje z anotačného súboru datasetu musia byť pred vstupom do neurónovej siete spracované do takej podoby aby z nich bolo možné vytvoriť „ground truth“ teda očakávaný výsledok siete, vzhľadom ku ktorému bude výstup siete porovnaný. Výsledkom tohto porovnania je chyba siete, ktorá ďalej slúži ako vstup optimalizéra. Presný popis implementácie je možné vidieť v kapitole 4.3. Predstavujeme teda formát BB3TXT inšpirovaný formátom BB3TXT [18], v ktorom sú uložené informácie, ktoré slúžia na korektné vytvorenie „ground truth“ siete. Údaje v súbore sú oddelené medzerou podobne ako v prípade anotácii KITTI. Informácie o pozícii ohraničujúceho kvádra sú uložené v súradnicovom systéme obrázka, čím je zabezpečené, že tieto údaje sú nezávislé na použitej kamere a datasete. Po rekonštrukcii kvádra do obrázka získame 8 vrcholov v súradnicovom systéme obrázka. Na spätnú projekciu však stačia 3 vrcholy na spodnej stene kvádra a jeden vrchol na hornej stene ako bolo spomenuté v kapitole 3.5. Vďaka tomu je možné redundantné údaje po projekcii ignorovať a využiť iba minimum z nich. Zvolený prístup odľahčí nielen samotnú sieť ale aj pamäťové nároky pri trénovaní.

Počet hodnôt	Príklad	popis
1	000704	Názov súboru
1	Car	Type (Typ) - objektu
1	754,029	Súradnica x bodu fbl
1	280,346	Súradnica y bodu fbl
1	864,929	Súradnica x bodu fbr
1	280,524	Súradnica y bodu fbr
1	864,057	Súradnica x bodu rbl
1	366,878	Súradnica y bodu rbl
1	164,670	Súradnica y bodu ftl
1	187,362	Súradnica x stredu ohraničujúceho kvádra
1	90,112	Súradnica y stredu ohraničujúceho kvádra
1	318	najväčšia hodnota z (výška/šírka objektu)

Tabuľka 4: Popis formátu BB3TXT

Na vstupe navrhnutej siete sú obrázky s veľkosťou (128×256), preto hodnoty stredu ohraničujúceho kvádra musia byť normalizované do tohto rozpätia ešte pred vstupom do siete, kde je k dispozícii pomer strán obrázka. Hodnoty vrcholových bodov musia ostať pôvodné. To sú hodnoty, ktoré chceme pomocou siete predikovať. Súradnice bodov nie sú normalizované do rozpätia vstupného obrázka pretože cieľom je sieť naučiť predikovať tieto hodnoty vzhľadom k reálnej veľkosti obrázka.

3.5 Projekcia z KITTI datasetu do snímok z kamery

Cieľom tejto práce je zistiť ohraničujúce kvádre áut v scéne z monokulárnej kamery. Táto úloha je nemožná bez ďalších informácii o scéne ako je napríklad hĺbka. Namiesto používania stereo kamier bol použitý predpoklad, že ohraničujúce kvádre ležia na základnej rovine. Z tohto predpokladu vyplýva, že na zistenie polohy kvádra stačí rovnica roviny, na ktorej kváder leží a projekčná matica obrázka, z ktorej vieme zistiť všetky potrebné údaje podľa kapitoly 3.7. Rovnicu roviny a jej normálu je možné získať z výstupu algoritmu v kapitole 3.3. Na rekonštrukciu kvádra stačia 3 vrcholy na základnej rovine X^I_{fbl}, X^I_{fbr} *a* X^I_{rbr} a y súradnica vrcholu X^I_{ftr} z hornej steny kvádra. Keďže KITTI anotácie obsahujú informácie v metroch bude nutné použiť obrazovú projekčnú maticu, ktorá je uvedená v kalibračných súboroch datasetu.

3.6 Získanie bodov v lokálnom súradnicovom systéme

Pred tým ako začne projekcia vrcholov kvádra do obrázka je nutné poznať ich hodnoty v lokálnom súradnicovom systéme. V anotáciach datasetu je uvedený stred spodnej steny kvádra ako pozícia objektu. V lokálnom súradnicovom systéme auta je možné chápať tento bod ako bod

[0, 0, 0] v 3D. Od tohto bodu vieme určiť všetky vrcholy kvádra tak ako je to naznačené nižšie. Ďalej je dôležité si uvedomiť, že podľa datasetu vzdialenosť objektu od kamery určuje súradnica x, nie z. Viď obr 5.

Príklad:
l - dĺžka objektu
w - šírka objektu
h - výška objektu

$$\begin{pmatrix} fbl & fbr & rbl & rbr & ftl & ftr & rtl & rtr \\ l/2 & l/2 & -l/2 & -l/2 & l/2 & l/2 & -l/2 & -l/2 \\ 0 & 0 & 0 & 0 & -h & -h & -h & -h \\ w/2 & -w/2 & w/2 & -w/2 & w/2 & -w/2 & w/2 & -w/2 \end{pmatrix}$$

Takto získané vrcholy musia byť ešte pred spracovaním prevedené do homogénnych súradníc pretože projekčná matica $\boldsymbol{P}$ má tvar 3×4. Kapitola 3.

3.7 Inverzná projekcia

Pomocou inverznej projekcie je možné získať súradnice bodu v súradniciach kamery zo súradníc bodu v obrázku. Výsledkom projekcie je lúč z kamery do bodu v obrázku. Tento lúč však obsahuje nekonečne veľa bodov a preto je potrebné poznať ďalšie informácie o polohe bodu. V tomto prípade bola využitá rovnica roviny. Presnú polohu bodu v 3D je možné získať vypočítaním priesečníku základnej roviny a získaného lúča z inverznej projekcie 3.8. Uvažujme bod $\boldsymbol{X}^W$ a pozíciu kamery $\boldsymbol{C}$ v 3D, vzhľadom na daný obrázok. Súradnice pozície kamery sú dôležité pri presnom určení polohy bodu v 3D. Cez tento bod je vedený lúč, pomocou ktorého je možné zistiť presnú polohu bodu. Výsledkom projekcie bodu $\boldsymbol{X}^W$ do kamery so stredom v $\boldsymbol{C}$, rotáciou $\boldsymbol{R}$ a kalibráciou $\boldsymbol{K}$ je

$$\lambda \boldsymbol{X}^I = \boldsymbol{KR}(\boldsymbol{X}^W - \boldsymbol{C}) = (\boldsymbol{KR} | - \boldsymbol{KRC}) \begin{pmatrix} X^W \\ 1 \end{pmatrix},$$

kde $\boldsymbol{R}$ a $\boldsymbol{K}$ sú matice tvaru 3×3, $\boldsymbol{C}$ a $\boldsymbol{X}^W$ sú vektory 3×1. Bod $\boldsymbol{X}^I = (u,\ v,\ 1)^T$, kde u a v sú súradnice $[x, y]$ bodu v obrázku, reprezentuje lúč z bodu $\boldsymbol{C}$ smerom $(\boldsymbol{X}^W - \boldsymbol{C})$. Týmto spôsobom je možné získať lúč cez hľadaný bod, ale aby bolo možné určiť presnú polohu bodu v 3D je dôležité poznať ďalšie informácie. Na získanie presnej polohy bodu $\boldsymbol{X}^W$ je nutné spočítať inverznú maticu ku $\boldsymbol{KR}$ a pozíciu kamery. Tieto informácie je možné ľahko odvodiť z obrazovej

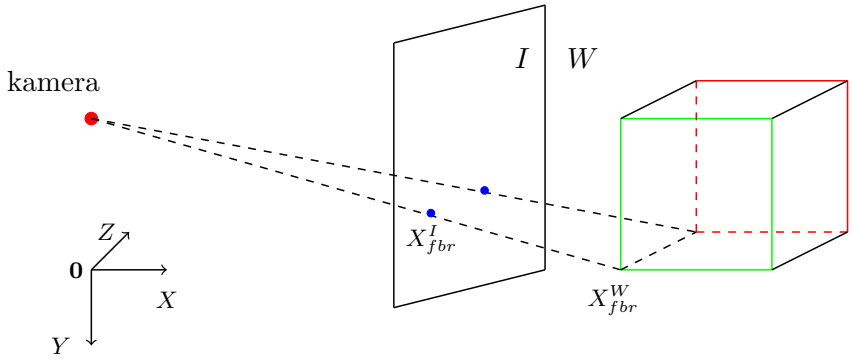
projekčnej matice $\boldsymbol{P} = \boldsymbol{KR}(\boldsymbol{I}| - \boldsymbol{C})$, ktorá je k dispozícii v kalibračných súboroch datasetu.

$$\begin{aligned} \lambda \boldsymbol{X}^I &= \boldsymbol{KRX}^W - \boldsymbol{KRC} \\ \boldsymbol{KRC} + \lambda \boldsymbol{X}^I &= \boldsymbol{KRX}^W \\ \boldsymbol{C} + \lambda (\boldsymbol{KR})^{-1} \boldsymbol{X}^I &= \boldsymbol{X}^W \end{aligned}$$

Výpočet inverznej matice ku **KR** je triviálny a stred kamery sa dá vypočítať nasledovne

$$\boldsymbol{C} = -(\boldsymbol{KR})^{-1} \boldsymbol{p}_4 \,,$$

kde $\boldsymbol{p}_4$ je štvrtý stĺpec projekčnej matice. Na obrázku 7 je zobrazený princíp inverznej projekcie, ktorá je využitá na získanie bodov na základnej rovine. Ako príklad je uvedený bod X^I_{fbr}. Ostatné body sú získané rovnako. Získané sú iba body na základnej rovine, kde vieme určiť ich presnú polohu podľa priesečníku so základnou rovinou, na ktorej je umiestnený každý ohraničujúci kváder 3.8. Body v hornej stene kvádra sú odvodené podľa kapitoly 3.9.



Obr. 7: Zobrazenie projekcie z obrázka do 3D. Na obrázku je zobrazený princíp inverznej projekcie, použitej na prevod súradníc z obrázka do súradníc kamery. I predstavuje oblasť zachytenú kamerou. W (world) predstavuje trojrozmernú oblasť (reálny svet), kde sa snažíme nájsť objekty

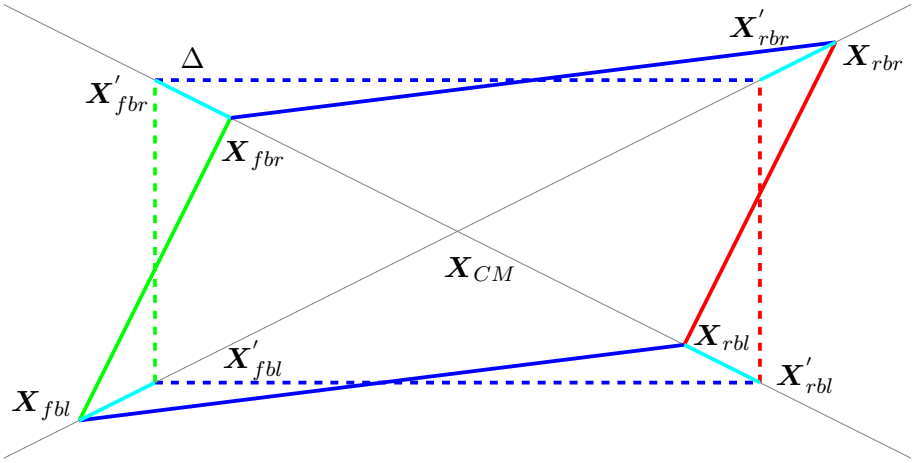
3.8 Rekonštrukcia plochy na základnej rovine

Na určenie ohraničujúceho kvádra bola použitá rovnica základnej roviny a inverzná projekcia. Ako bolo spomenuté v kapitole 3.5 budú stačiť 3 vrcholy na zistenie spodnej steny kvádra. Pre jednoduchosť bude použitý vrchol $\boldsymbol{X}^I$, ktorý reprezentuje jeden z vrcholov na základnej rovine. Z inverznej projekcie bodu $\boldsymbol{X}^I$ bol získaný lúč $\boldsymbol{I_{XW}} = \boldsymbol{C} + (\boldsymbol{KR})^{-1}\boldsymbol{X}^I$ prechádzajúci cez $\boldsymbol{C}$ a $\boldsymbol{X}^W$. Ak $\boldsymbol{X}^W$ leží na základnej rovine musí spĺňať $\boldsymbol{n} \cdot \boldsymbol{X}^W + d = 0$. Výsledkom po dosadením

za $\boldsymbol{X}^W$ je:

$$\begin{aligned}\boldsymbol{n} \cdot (\boldsymbol{C} + \lambda \boldsymbol{I}_{\boldsymbol{X}^I}) + d &= 0 \\ \boldsymbol{n} \cdot \boldsymbol{C} + \lambda \boldsymbol{n} \cdot \boldsymbol{I}_{\boldsymbol{X}^I} + d &= 0 \\ \lambda &= -\frac{\boldsymbol{n} \cdot \boldsymbol{C} + d}{\boldsymbol{n} \cdot \boldsymbol{I}_{\boldsymbol{X}^I}}\end{aligned}$$

Týmto spôsobom budú získané všetky vrcholy $\boldsymbol{X}^W_{fbl}$, $\boldsymbol{X}^W_{fbr}$ a $\boldsymbol{X}^W_{rbl}$ na základnej rovine. Chýbajúci vrchol bol dopočítaný ako $\boldsymbol{X}^W_{rbr} = \boldsymbol{X}^W_{fbr} + (\boldsymbol{X}^W_{rbl} - \boldsymbol{X}^W_{fbl})$. Keďže projekcia pozostáva iba z troch vrcholov výsledkom nie je pravouhlý obdĺžnik ale paralelogram.



Obr. 8: Výsledok rekonštrukcie spodnej steny ohraničujúceho kvádra na základnej rovine. X je vektor tvaru 3×1 a reprezentuje vrcholy na základnej rovine v 3D

Príklad ako môže byť paralelogram upravený do požadovaného stavu je zobrazený na obr. 8. Dôležité je zachovať stred objektu $\boldsymbol{X}_{CM}$. Dĺžka diagonál v obdĺžniku je rovnaká narozdiel od paralelogramu. Podľa tejto informácie je možné diagonály paralelogramu upraviť tak aby boli rovnaké. Úpravu diagonál je možné spraviť rôznymi spôsobmi, v tejto práci je použitý priemer diagonál ako ich referenčná dĺžka.
Použitím nižšie uvedených rovníc je možné získať nové súradnice vrcholov:

$$\begin{aligned}\boldsymbol{d}_1 &= \boldsymbol{X}_{fbl} - \boldsymbol{X}_{CM} \\ \boldsymbol{d}_2 &= \boldsymbol{X}_{fbr} - \boldsymbol{X}_{CM} \\ \Delta &= \frac{||\boldsymbol{d}_1|| - ||\boldsymbol{d}_2||}{2} \\ \boldsymbol{X}'_{fbl} &= \boldsymbol{X}_{CM} + \boldsymbol{d}_1(1 - \frac{\Delta}{||\boldsymbol{d}_1||}) \\ \boldsymbol{X}'_{fbl} &= \boldsymbol{X}_{CM} + \boldsymbol{d}_2(1 - \frac{\Delta}{||\boldsymbol{d}_2||})\end{aligned}$$

Podobným spôsobom boli získané vrcholy $\boldsymbol{X}^{'}_{rbl}$ a $\boldsymbol{X}^{'}_{rbr}$.

3.9 Rekonštrukcia steny kvádra nad objektom

Po zrekonštruovaní spodnej plochy je možné odvodiť vrcholy v ploche nad objektom. K tomu je nutné poznať aspoň jeden vrchol v danej rovine. Konkrétny vrchol bol odvodený ako x súradnica bodu, ktorý sa nachádza v priesečníku priamky, ktorá je kolmá na základnú rovinu, zo základnou rovinou a y súradnica popísaná v kapitole 3.5. V predošlej kapitole je napísané, že spodná stena kvádra je pravouhlý obdĺžnik. Vďaka tejto informácii je možné použiť vektor daný bodmi $\boldsymbol{X}^{W}_{fbl}$ a $\boldsymbol{X}^{W}_{rbl}$ ako normálový vektor prednej steny kvádra $\boldsymbol{n}_F = (a_F, b_F, c_F)$ a dosadiť $\boldsymbol{X}^{W}_{fbl}$ do rovnice prednej steny

$$\boldsymbol{n}_F = \boldsymbol{X}^{W}_{fbl} - \boldsymbol{X}^{W}_{rbl}$$
$$d_F = -(\boldsymbol{n}_F \cdot \boldsymbol{X}^{W}_{fbl})$$

Na základe týchto vzťahov bola odvodená rovnica roviny prednej steny ako $a_F x + b_F y + c_F z + d_F = 0$. Pri hľadaní priesečníku prednej steny a lúča $\boldsymbol{X}^{W}_{ftl} = \boldsymbol{C} + (\boldsymbol{KR})^{-1}\boldsymbol{X}^{I}_{ftl}$ cez bod $\boldsymbol{X}^{I}_{ftl}$ sa postupuje rovnako ako pri získavaní bodov na základnej rovine. Výsledkom je bod v 3D v metroch. Z výsledku je možné zistiť výšku kvádra a zrekonštruovať tak plochu nad objektom. Keďže spodná stena kvádra má y súradnicu 0 v lokálnom súradnicovom systéme stačí pripočítať zistenú výšku k už známym bodom na základnej rovine.

4 Implementácia neurónovej siete

Neurónové siete je možné si predstaviť ako projekciu z N-rozmerného priestoru $X^N \subset \mathbb{R}^N$ do M-rozmerného priestoru $Y^M \subset \mathbb{R}^M$. Priestor Y^M môžme jednoducho popísať ako zreťazenie lineárnych a nelineárnych funkcii $l_i : X^{N_i} \rightarrow Y^{M_i}$ nazývaných vrstvy. Obecne môžme tento vzťah zapísať ako

$$y = (l_L \circ ... \circ l_2 \circ l_1)(x),\ y \in Y^M,\ x \in X^N,\ L \in \mathbb{N}\,.$$

Skryté konvolučné vrstvy pozostávajú z neurónov. Každý neurón predstavuje jednu elementárnu operáciu, ktorú je možné zapísať nasledovne

$$l_{ij}(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i \cdot \boldsymbol{w}_{ij} + b_j)\,,$$

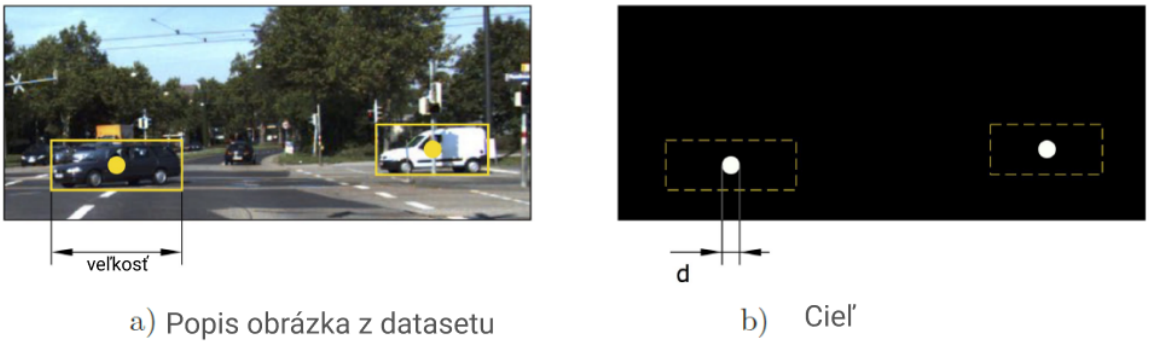
kde w_{ij} predstavuje váhy neurónu, ktoré sa aplikujú na vstupný vektor, b_j je bias, ktorý v prípade neurónových sietí slúži na posun aktivačnej funkcie [9] prípadne na zvýšenie variability váh danej vrstvy, čoho výsledkom je obecnejší model a presnejší výstup. ϕ je aktivačná funkcia, v tomto prípade *Relu*. Operácia $l_{ij}(\boldsymbol{x}_i)$ je aplikovaná na každý neurón j v každej vrstve i.

4.1 Tensorflow framework

TensorFlow (TF) [1] predstavuje „end-to-end“ systém pre vytváranie, testovanie a nasadenie ML modelov na rôzne zariadenia. TF ponúka základné rozhranie pre všetky potrebné funkcie a algoritmy, ktoré sú bežne využívané pri ML. Ponúka však aj API rozhranie, ktoré zapuzdruje jednotlivé funkcie do celkov a zrýchľuje a uľahčuje tak prácu programátorov. API rozhranie nesie názov Keras podobne ako rovnomenný ML framework. Keras framework je možné použiť aj bez TF. TF postupne integrovalo Keras framework a s príchodom TF 2.x vzniklo plnohodnotné API rozhranie Keras, ktoré je takmer identické so samostatným frameworkom. Jednou z výhod TF je jednoduchý a prehľadný „debugger“. Pre maximálne využitie dostupných zdrojov ponúka TF funkcie pre distribuované učenie („distributed strategy“), čo znamená rozdelenie záťaže na viacero procesorov alebo viacero grafických kariet. TF predstavuje niekoľko verzií. Samostatné TF, TF Lite pre mobilné zariadenia a TF.js pre webové prehliadače. Pre korektné nasadenie aplikácie a jej kontrolu slúži nadstavba TFX. Framework bol s príchodom verzie 2.x prerobený takmer od základov. Keďže na začiatku práce sme začínali s TF 1.x a neskôr sme prešli na verziu 2.1 niekoľko implementačných častí sa muselo zmeniť. Tieto zmeny sú popísané priamo v texte tam, kde nastali zmeny oproti verzii TF 1.x. V kapitole 6.1 sú popísané možnosti ako je možné zrýchliť trénovanie s novou verziou TF.

4.2 Vytvorenie referenčných dát

Pixely vo výstupe siete v určitom okruhu $r = d/2$ od stredu objektu (od stredu 2D ohraničujúceho štvoruholníka) obsahujú súradnice bodov daného objektu podľa obrázka 3. Práca je zameraná na detekciu 3D ohraničujúcich kvádrov, z tohto dôvodu výsledok obsahuje súradnice bodov $\boldsymbol{X}_{fbl}$, $\boldsymbol{X}_{fbr}$, $\boldsymbol{X}_{rbl}$ a y súradnicu bodu $\boldsymbol{X}_{ftl}$.



Obr. 9: Stred objektu je odvodený z KITTI datasetu podľa 2D ohraničujúceho štvoruholníka takto $C_x = \frac{tl_x + br_x}{2}$, $C_y = \frac{tl_y + br_y}{2}$. Obrázok *b* znázorňuje pravdepodobnostnú mapu výstupu siete, kde stred objektu má predpokladanú hodnotu 1, okolie 0. Veľkosť stredu a jeho blízkeho okolia závisí od použitej mierky [18]

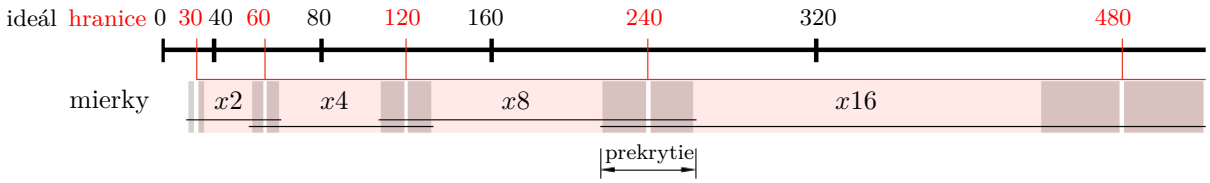
Veľkosť okruhu od stredu objektu, ktorý slúži na detekciu objektu je ľubovoľná. Ideálna veľkosť objektov, ktoré budú detegované danou mierkou bola odvodená od veľkosti okolia a parametra *cr*. Parameter *cr* je možné zvoliť ľubovoľne, no v práci bola pre všetky mierky použitá hodnota $cr = 0,25$. Výsledkom je

$$max\_dim = \frac{2 \cdot r + 1}{cr} \cdot s\,,$$

dosadením r a cr je možné získať veľkosť objektu, ktorý bude detegovaný v danej mierke. Napríklad pre $cr = 0,25$ a $r = 2$ dostaneme veľkosť objektu $max\_dim = 80$ pre mierku $s = 4$, čo znamená, že výstup siete pre mierku 4 by mal detegovať objekty veľkosti okolo 80 pixelov [18]. Je dôležité si uvedomiť, že 80 pixelov v tomto prípade je **ideálna veľkosť objektu**, ktorá by mala byť detegovaná mierkou $s = 4$ avšak v realite by bolo nutné vytvoriť výstup pre všetky možné veľkosti objektov. Preto bol pre každú mierku vytvorený rozsah veľkostí, ktorý by mala detegovať.
Použitie mierok (2, 4, 8, 16) vychádza z faktu, ža každá „pooling“ vrstva, ktorá sa nachádza za každým výstupom zo siete okrem posledného, zredukuje výšku a šírku obrázka na polovicu. Každá mierka bola prispôsobená tak aby detegovala objekty v určitom rozsahu podľa obrázka 10. Rozsah veľkostí pre jednotlivé mierky je možné vypočítať nasledovne

$$\left[\frac{max\_dim_i + max\_dim_{i-1}}{2} - o_i^L, \frac{max\_dim_i + max\_dim_{i+1}}{2} + o_i^R\right],$$

kde o_i^L je ľavé prekrytie mierok a o_i^R pravé. Prekrytie slúži na odstránenie skokovej zmeny medzi mierkami a veľkosťami objektov, ktoré detekujú. Takto navrhnuté mierky a ich rozsah spôsobí že niektoré objekty môžu byť detegované v dvoch mierkach súčasne.



Obr. 10: Distribúcia veľkostí objektov podľa rozsahu jednotlivých mierok ($cr = 0,25$, $r = 2$). Každá mierka detekuje objekty danej veľkosti plus objekty, ktoré sa môžu prekrývať so susednými mierkami. Zdroj [18]

Pre konkrétnu mierku musia byť vybrané len objekty, ktoré svojou veľkosťou odpovedajú rozsahu pre danú mierku. Funkcia vo výpise 1 vypočíta veľkosť objektov, pre ktoré budú vytvorené referenčné dáta. Z anotácii datasetu budú vybrané iba objekty, ktoré podľa svojej veľkosti patria do danej mierky.

```
def GetObjectBounds(radius, cr, boundaries, scale):
    ideal_size = (2.0 * radius + 1.0) / cr * scale
    ext_above = ((1.0 - boundaries) * ideal_size) / 2.0 + boundaries *
        ideal_size
    bound_above = ideal_size + ext_above
    diff = ideal_size / 2.0
    ext_below = ((1 - boundaries)* diff) / 2.0 + boundaries * diff
    bound_below = ideal_size - ext_below

    return bound_above, bound_below, ideal_size
```

Výpis 1: Výpočet veľkosti objektov pre konkrétnu mierku

4.2.1 Úprava referenčných dát pomocou Gausovho filtra

V prípade prvej vrstvy cieľa, ktorá predstavuje pravdepodobnostnú mapu nájdených objektov, je dôležité aby pixely v okolí stredu objektu mali hodnotu 1 a ostatné 0. Výsledkom takéhoto postupu by bola náhla zmena hodnoty pixelov na hrane stredu objektu, čo vedie k prudkému

nárastu chyby na danom objekte ak by výsledok siete v tomto bode bol len minimálne posunutý. Použitie Gausovho filtra odstráni tieto nedostatky a zabezpečí rýchlejšie (presnejšie) učenie siete. Na pravdepodobnostnú mapu cieľa bol použitý gausov filter s veľkosťou 3×3 a $\sigma = 1$. Použitie gausovho filtra má za následok plynulý prechod medzi detegovaným objektom a jeho okolím. V prípade, že by detegovaný objekt na výstupe siete bol len minimálne posunutý chyba na tomto objekte nebude taká významná ako v prípade bez gausovho filtra. Pre $r = 2$ vyzerá výstup upravený gausovým filtrom nasledovne

(a) Výstup siete (b) Ground truth (c) Chyba siete

Obr. 11: Výsledok siete pre mierku 8 (a), ground truth podľa anotácii z datasetu KITTI (b), chyba siete (c). Hodnoty na obrázkoch (a), (b), (c) sú normalizované do intervalu $\langle 0,\, 255\rangle$, aby boli lepšie viditeľné. Referenčné dáta sú tvorené pomocou OpenCV funkcie *circle*, ktorá pri malej hodnote r nevytvorí kruh ale štvorec. Tento fakt však nemá vplyv na trénovanie a preto bol zanedbaný.

Vrstvy 1 až 7, ktoré obsahujú súradnice bodov ohraničujúceho kvádra musia byť tiež upravené a normalizované. Súradnice sa nachádzajú pod každým pixelom (pixel je možné chápať ako pole s ôsmimi prvkami, keďže výstup obsahuje 8 vrstiev), ktorý má v pravdepodobnostnej vrstve hodnotu väčšiu ako 0. Každý pozitívny pixel z pravdepodobnostnej mapy predstavuje jeden ohraničujúci kváder. Hodnoty súradníc sú normalizované do rozsahu $\langle 0,\, 1\rangle$ pretože to je vhodnejšie pre sieť v priebehu trénovania a dosiahne sa tak vyrovnanie gradientu z pravdepodobnostnej vrstvy a vrstiev nesúcich hodnoty súradníc. Hodnoty súradníc sú normalizované nasledovne pre vrstvy, ktoré obsahujú x súradnice bodov

$$v' = 0,5 + \frac{v - x - j \cdot scale}{ideal}\,,$$

pre vrstvy, ktoré obsahujú y súradnice bodov

$$v' = 0,5 + \frac{v - y - l \cdot scale}{ideal}\,,$$

kde v predstavuje hodnotu danej súradnice, *ideal* je ideálna veľkosť objektu vzhľadom na mierku v ktorej sa objekt nachádza, *scale* je mierka, pre ktorú je cieľ vytvorený. $[x,\, y]$ sú súradnice stredu daného objektu podľa formátu BB3TXT 3.4. V prípade 1. (prob) vrstvy bol použitý gausov filter aby prechod medzi nulovými a nenulovými pixelmi bol plynulejší. Gausov filter nie je možné použiť na vrstvy so súradnicami, pretože hodnoty po aplikovaní filtra by neodpovedali hodnotám, ktoré chceme, aby sa sieť naučila. Bolo nutné zaistiť, aby hodnoty súradníc mimo stredu

objektu v intervale $\langle -r, r \rangle$ boli upravené tak, aby zohľadňovali svoju vzdialenosť od stredu. K tomu slúžia premenné j a l, ktoré sú v intervale $\langle -r, r \rangle$ od stredu objektu. Presná implementácia je uvedená vo výpise 2. Na ukážke kódu je zobrazená iba časť pre vrstvy so súradnicami, vytvorenie prvej *prob* je jednoduché s využitím OpenCV. Implementácia sa po prechode na TF 2.x nezmenila, iba sa presunula zo stratovej funkcie do časti pred trénovaním.

```
for c in range(1,8):
    for l in range(-r,r,1):
        for j in range(-r,r,1):
            xr = x + j
            yr = y + l
            if xr >= 0 and xr < width and yr >= 0 and yr < height:
                if maps[0][yr][xr] > 0.0:
                    if c ==1 or c == 3 or c == 5:
                        maps[c][yr][xr] = 0.5 + (target[c-1] - x - j * scale)
                            / ideal
                    elif c == 2 or c == 4 or c == 6 or c == 7:
                        maps[c][yr][xr] = 0.5 + (target[c-1] - y - l * scale)
                            / ideal
```

Výpis 2: Vytvorenie referenčných dát

kde c označuje niektorú z vrstiev výstupu, ktoré obsahujú informácie o súradniciach jednotlivých bodov podľa 3. *width* a *height* definujú šírku a výšku všetkých vrstiev pre danú mierku. *target* je vo formáte BB3TXT a obsahuje informácie o objektoch na obrázku.

4.3 Implementácia stratovej funkcie (loss function)

Výstup navrhnutej siete obsahuje 8 vrstiev, z ktorých prvá predstavuje pravdepodobnosť nájdeného objektu (spokojnosť siete s nájdeným objektom), ostatné obsahujú údaje o vrcholoch kvádra z ktorých bude možné zrekonštruovať celý ohraničujúci kváder. Výstup je popísaný v kapitole 2.3. Výstup zo siete je nutné s niečím porovnať aby sme získali chybu siete, ktorá bude ďalej použitá na výpočet gradientu pri optimalizácii. Na vstupe väčšiny konvolučných neurónových sietí je obraz a požadovaný výstup (ďalej ako cieľ). Cieľ slúži ako vstup stratovej funkcie, ktorá porovná skutočný výstup zo siete a cieľ. V našom prípade by cieľ musel obsahovať 8 vrstiev pre každú mierku (scale). Takto vytvorený cieľ by bol na vstupe veľmi neefektívny, preto boli ako vstup použité údaje z formátu BB3TXT. Stratová funkcia je založená na kvadratickej Euklidovskej vzdialenosti.

$$E = \frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2\,,$$

kde t_i je vrstva z cieľa, y_i je vrstva z výstupu siete a i prechádza všetky neuróny výstupnej vrstvy.
Pre jednoduchosť popíšeme stratu iba pre jeden obrázok a pre jednu výstupnú mierku. Vrstva $c = 0$ je pravdepodobnostná vrstva, ostatné sú vrstvy obsahujúce konkrétne súradnice bodov. Uvažujme t_i^c ako cieľ a y_i^c ako skutočný výstup zo siete, kde c predstavuje vrstvu výstupu ($c \in 0, ..., 7$) a i konkrétny pixel v danej vrstve. Upravená stratová funkcia:

$$N_p = \sum_{i=1}^{N}[\![t_i^0 \neq 0]\!]$$

$$E = \frac{1}{2N}\sum_{i=1}^{N}(t_i^0 - y_i^0)^2 + \frac{1}{2N_p(C-1)}\sum_{i=1}^{N}\sum_{c=1}^{C-1} t_i^0(t_i^c - y_i^c)^2\,,$$

kde N_p je počet pozitívnych pixelov v prvej (pravdepodobnostnej) vrstve cieľa, $[\![x]\!]$ je Iversonova notácia, ktorá má na výstupe 0 alebo 1 podľa špecifikovanej podmienky, C je počet výstupných vrstiev. Časť so súradnicami bodov je navyše vynásobená hodnotou z pravdepodobnostnej mapy čo v prípade Gausiánu na výstupe zníži stratu na pixeloch, ktoré sú ďalej od stredu objektu.
Problémom vyššie spomenutej stratovej funkcie je nerovnováha medzi pozitívnymi a negatívnymi pixelmi vo výstupe. Počet pozitívnych pixelov $N_p << N_N$ kde $N_N = N - N_p$ je oveľa menší ako počet negatívnych pixelov. Gradient z negatívnych pixelov by prevažoval na gradientom z pozitívnych pixelov. Z tohto dôvodu bola stratová funkcia upravená tak aby sa gradienty aspoň čiastočne vyrovnali

$$E = \frac{1}{2N}\sum_{i=1}^{N}(1 + [\![t_i^0 \neq 0]\!](\alpha - 1))(t_i^0 - y_i^0)^2 + \frac{1}{2N_p(C-1)}\sum_{i=1}^{N}\sum_{c=1}^{C-1} t_i^0(t_i^c - y_i^c)^2$$

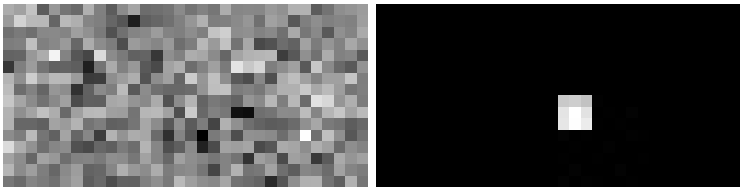
V niektorých prípadoch môže dôjsť k deleniu nulou. V druhej časti rovnice, ktorá počíta chybu na vrstvách 1 až 7 v prípade ak by referenčné dáta neobsahovali ani jeden objekt N_p bude 0. V prípade ak $N_p = 0$ výpočet v druhej časti rovnice sa preskočí a celá táto časť sa nahradí 0. Vyššie uvedená stratová funkcia je definovaná pre jeden obrázok. Pri spracovaní viacerých obrázkov je nutné vytvoriť cieľ podľa kapitoly 4.2 pre každý obrázok a stratovú funkciu aplikovať na každý z nich. Výsledok porovnania je interpretovaný ako chyba na danom obrázku. Výsledok

stratovej funkcie je suma chýb na všetkých obrázkoch zo vstupu. Sieť detekuje objekty v rôznych mierkach (2, 4, 8, 16), takto navrhnutá stratová funkciu musí byť spustená pre každú mierku samostatne. Suma výsledkov zo stratovej funkcie pre každú mierku tvorí vstup optimalizéra. Stratová funkcia pre Tf 1.x je bežná funkcia. V prípade TF 2.x stratovú funkciu predstavuje objekt odvodený z abstraktnej triedy *tensorflow.keras.losses.Loss*. Objekt tejto triedy je tzv. „callable“. Trieda obsahuje metódu *call*, ktorá sa automaticky vykoná po zavolaní objektu a vypočíta chybu siete. Príklad takejto funkcie je uvedený vo výpise 4.

```
def compute_loss(target, output, scale):
   error_sum = 0
   for i in range(N):
      o_i = output[i]
      t_i = target[i]
      gt = create_ground_truth(t_i, scale)
      error = loss(gt, o_i)
      error_sum += error

   return error_sum
```

Výpis 3: Implementácia stratovej funkcie pre TF 1.x

Vstup funkcie 3 tvoria 3 parametre. *scale* predstavuje mierku (2, 4, 8, 16), pre ktorú bude loss funkcia počítať chybu siete. *output* predstavuje skutočný výstup siete. *target* predstavuje informácie o objektoch podľa formátu BB3TXT 3.4. Parameter *target* obsahuje informácie pre všetky objekty na obrázku, teda pre všetky mierky. Funkcia *create_ground_truth* berie ako druhý argument mierku (*scale*), podľa ktorej vyberie len tie objekty, ktoré podľa ich veľkosti patria do danej mierky. Dôležité je, aby nedošlo z zámene výstupu siete a mierky. Ak by parameter *output* predstavoval informácie pre mierku 2 a parameter *scale* by bol 4 funkcia *create_ground_truth* by vytvorila „ground truth“ s objektami, ktoré patria do mierky 4. Takáto zámena parametrov by spôsobila, že chyba siete pre mierku 2 by sa prestala znižovať a začala by kolísať v okolí nejekej hodnoty a samotný výstup siete by bol nepoužiteľný, pretože by obsahoval hodnoty, ktoré neodpovedajú hodnotám podľa BB3TXT formátu.



(a) (b)

Obr. 12: Výsledok siete pre obrázok 000003.*jpg* s mierkou 8. Na obrázku (a) je zobrazený výstup so zle nastavenými parametrami loss funkcie, (b) zobrazuje výsledok pre rovnaký obrázok s dobre nastavenými parametrami

Funkcia *create_ground_truth* musí byť v prípade TF 1.*x* zavolaná vrámci funkcie *tensorflow.py_function* alebo *tensorflow.numpty_function*. Takéto volanie zabezpečí, že daná funkcia nebude súčasťou výpočtového grafu, ktorý si TF vytvára na pozadí a bude možné použiť balíček OpenCV na vytvorenie referenčných dát „ground truth“. Takto vytvorená stratová funkcia má niekoľko výhod. Jednou z výhod je minimum prenesených údajov, pretože stratová funkcia príma parameter *target* vo formáte BB3TXT, ktorého veľkosť je približne 36 B. Druhou výhodou je, že *gt* je vytvorený až v priebehu výpočtu a funkcia tak nie je veľmi pamäťovo náročná. Ak by sme ako parameter posielali už vytvorený *gt* pamäťové a prenosové nároky by sa rapídne zvýšili približne na 347 kB pre každý obrázok. Po prechode na TF 2.*x* musela byť stratová funkcia upravená tak aby neobsahovala volanie cez vyššie spomenuté funkcie, pretože spôsobujú úniky pamäte („memory leaks“)[1]. V tomto prípade už nie je možné sa vyhnúť zvýšenému prenosu údajov. V stratovej funkcii pre TF 2.*x* parameter *target* už nie je vo formáte BB3TXT ale predstavuje už vytvorený *gt* pre danú mierku (scale). Implementácia stratovej funkcie pre TF 2.*x* je uvedená vo výpise 4.

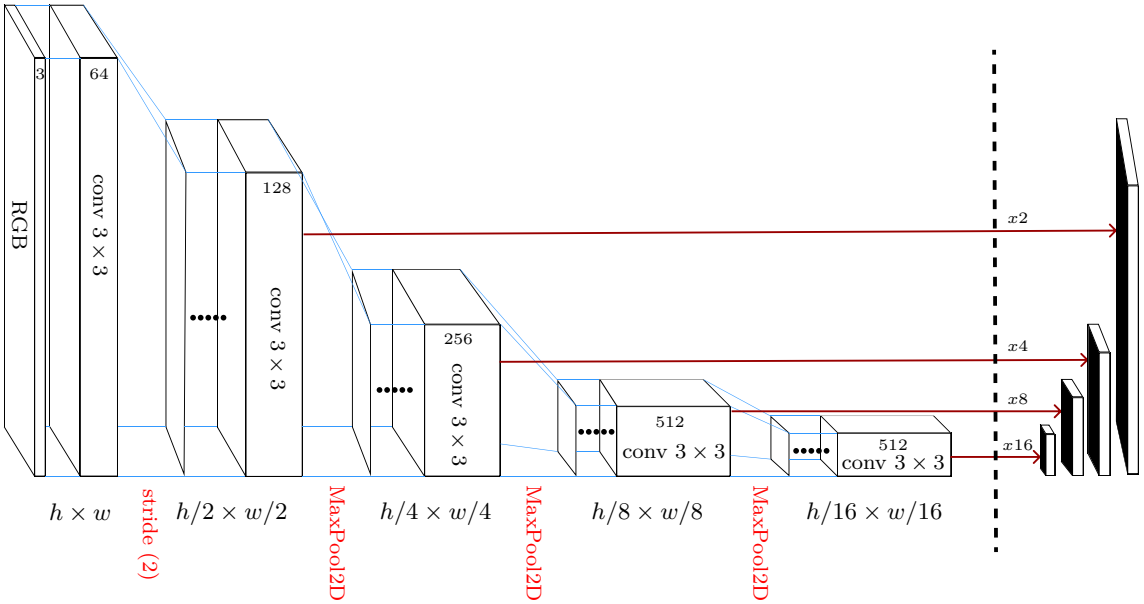
```
def call(target, output):
   error_sum = 0
   for i in range(N):
      o_i = output[i]
      t_i = target[i]
      error = loss(t_i, o_i)
      error_sum += error

   return error_sum
```

Výpis 4: Implementácia stratovej funkcie pre TF 2.*x*

[1] https://github.com/tensorflow/tensorflow/issues/35010

4.4 Archutektúra siete

Dizajn neurónovej siete v tejto práci je inšpirovaný neurónovou sieťou [18]. Sieť bola upravená tak, aby detegovala iba 3D ohraničujúce kvádre automobilov z RGB obrázkov. Výstup siete je založený na metóde DenseBox 2.3, ktorej výstup bol upravený tak, aby obsahoval všetky potrebné údaje pre inverznú projekciu. Použitie plne konvolučnej architektúry umožnilo detekciu objektov rôznych veľkostí. Princíp takejto siete je v posúvaní pomyselného okna (sliding window) po vstupe. Výsledkom každého posunutia je pravdepodobnosť, že hľadaný objekt sa nachádza presne v strede tohto okna. Každý pixel na výstupe predstavuje jeden objekt s nejakou pravdepodobnosťou 3.2.



Obr. 13: Zloženie siete podľa [18]. Na vstupe je RGB obrázok, ktorý je postupne upravený podľa použitej mierky.

Na obrázku 13 je zobrazená kompletná architektúra siete. Sieť deteguje objekty v mierkach (2, 4, 8, 16) ako je to naznačené za prerušovanou čiarou. Jednotlivé výstupy sú extrahované v priebehu prechodu sieťou tak aby zachytávali objekty definovanej veľkosti. Tieto údaje nemajú ďalej vplyv na trénovanie a sieť pokračuje s dátami z predchádzajúcej vrstvy. Ukážka, čo sa stane ak by sieť pokračovala v trénovaní s výstupnou vrstvou je popísaný v kapitole 6.2. Medzi jednotlivými časťami siete, ktoré sú zodpovedné za detekciu objektov v danej mierke je použitá „MaxPool2D“ vrstva. Táto vrstva zmenší veľkosť obrázka o polovicu, čo výrazne zmenší počet parametrov, ktoré sa sieť musí naučiť. Vrstva tiež spôsobí zväčšenie FOV 4.4.2 a teda zväčšenie veľkosti detegovaných objektov. Vrstvu „MaxPool2D“ je možné nahradiť konvolúciou s parametrom $stride = 2$. Práca Jost Tobias Springenberga [25] ukázala, že nahradenie „MaxPool2D“

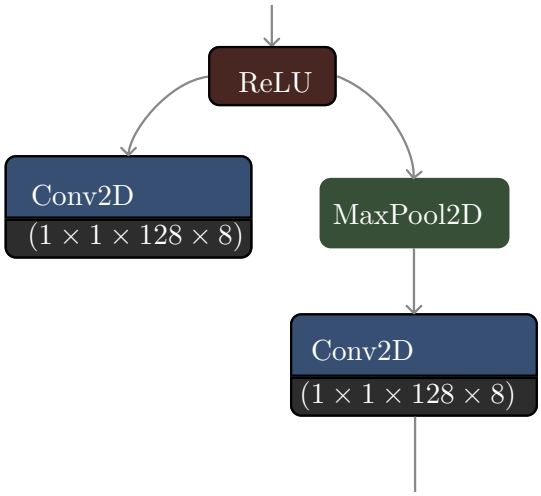
vrstiev môže čiastočne zvýšiť presnosť siete. Upravená architektúra a porovnanie výstupov je popísaný v kapitole 6.3. Podrobný popis vyššie zobrazenej architektúry je v tabuľke 5.

	Veľkosť filter	„stride“	Počet filtrov	FOV (px)	Ideál (px)	Veľkosť objektov (px)
1	3×3, $d = 1$	1	64	3×3		
2	3×3, $d = 3$	2	64	9×9		
3	3×3, $d = 1$	1	128	14×14		
4	3×3, $d = 2$	1	128	22×22		
5	3×3, $d = 4$	1	128	38×38		
6	3×3, $d = 7$	1	128	66×66		
7	1×1	1	8	66×66	33×33	$\langle 22.5, 55.5 \rangle$
8	„MaxPool2D“					
9	3×3, $d = 1$	1	256	76×76		
10	3×3, $d = 2$	1	256	92×92		
11	3×3, $d = 4$	1	256	124×124		
12	1×1	1	8	124×124	66×66	$\langle 44.5, 111 \rangle$
13	„MaxPool2D“					
14	3×3, $d = 1$	1	512	138×138		
15	3×3, $d = 2$	1	512	162×162		
16	3×3, $d = 4$	1	512	210×210		
17	1×1	1	8	210×210	133×133	$\langle 89, 222 \rangle$
18	„MaxPool2D“					
19	3×3, $d = 1$	1	512	228×260		
20	3×3, $d = 2$	1	512	260×260		
21	3×3, $d = 4$	1	512	324×324		
22	1×1	1	8	324×324	266×266	$\langle 178, 444 \rangle$

Tabuľka 5: Detailný popis architektúry siete, ktorá deteguje objekty v mierke (2, 4, 8, 16).

Tabuľka 5 zobrazuje presný popis architektúry siete a použitých parametrov. Sieť je rozdelená na 4 časti. Jednotlivé časti predstavujú mierky („scale“) objektov, ktoré budu v nich detegované. Výstup siete je extrahovaný z konvolučnej vrstvy 1×1 na konci každej časti. Je dôležité upozorniť, že táto vrstva nemá vplyv na ďalšie spracovanie a nasledujúca vrstva „MaxPool2D“ pracuje s výsledkom predchádzajúcej konvolučnej vrstvy. Vstup 8. vrstvy tvorí výstup 6. vrstvy a nie je tak ovplyvnený vrstvou 7. Podobne pre vrstvy 13 a 18. Obrázok 14 graficky znázorňuje časť siete kde dochádza k takémuto rozdeleniu. Vrstva 2 využíva parameter „stride“ na redukciu vstupu ako je to popísané v kapitole 4.4.2. Takáto úprava zníži pamäťové nároky siete. So zväčšujúcou sa mierkou sa zvyšuje aj počet konvolučných filtrov. Posledná časť siete využíva rovnaký počet filtrov ako predchádzajúca časť. V prípade, že by sa počet parametrov zvýšil na

1024 počet parametrov siete by presiahol 130 MB. Ak zanedbáme toto zvýšenie sieť obsahuje iba 60 MB parametrov. Podľa práce [18] zvýšenie filtrov v poslednej časti predstavuje len minimálne zlepšenie vo výsledkoch a vzhľadom na pamäťovú náročnosť je toto zlepšenie zanedbateľné.



Obr. 14: Grafické zobrazenie prechodu medzi mierkami v sieti. „Relu“ predstavuje výstup predchádzajúcej vrstvy, v tomto prípade vrstvy 6. Ľavá vetva grafu predstavuje výstupnú vrstvu, v tomto prípade výstup pre mierku 2. Výstup tejto vrstvy má tvar (8, 64, 128) a obsahuje informácie podľa 2.3. Pravá vetva grafu pokračuje v spracovaní pre ďalšie mierky

4.4.1 Vstupná vrstva

Vstupná vrstva siete príma ako vstup RGB obrázok ľubovoľnej veľkosti. Úlohou vstupnej vrstvy je pretypovanie hodnôt na dátový typ, ktorý sa používa v celom modely. Ďalšou úlohou vstupnej vrstvy je normalizácia hodnôt jednotlivých pixelov $\langle 0,\, 255\rangle$ do intervalu $\langle -1,\, 1\rangle$, čo je vhodnejšie počas trénovania. Hodnoty sú normalizované nasledovne

$$\frac{v - min(v)}{max(v) - min(v)}\,,$$

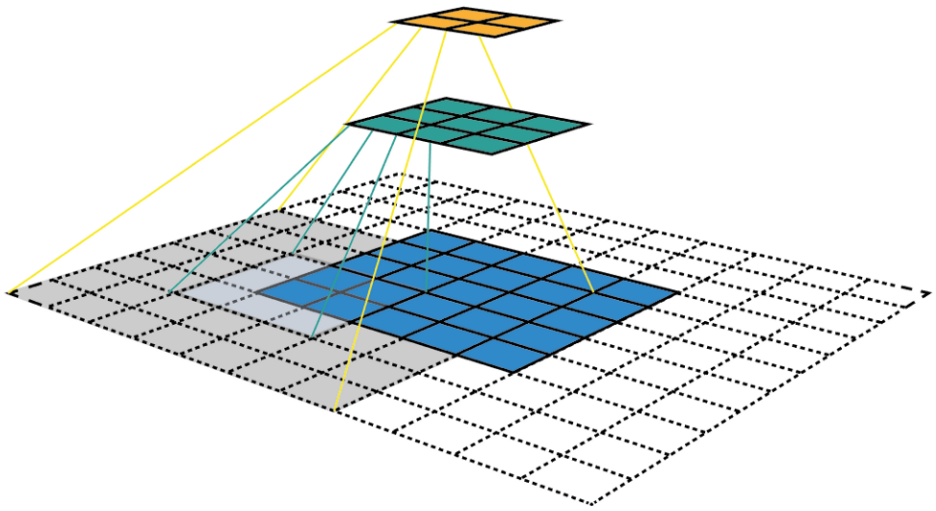
kde v sú pôvodné hodnoty pixelov. Počas trénovania sú obrázky z datasetu náhodne vyberané podľa hodnoty „batch size“ a možnosti zariadenia, na ktorom prebieha trénovanie.

4.4.2 Konvolučná vrstva

Konvolúcia je matematický operátor, ktorý spracováva dve funkcie f a g a jeho výsledkom je tretia funkcia $(f * g)$ definovaná ako

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f(m) \cdot g(n - m)\,,$$

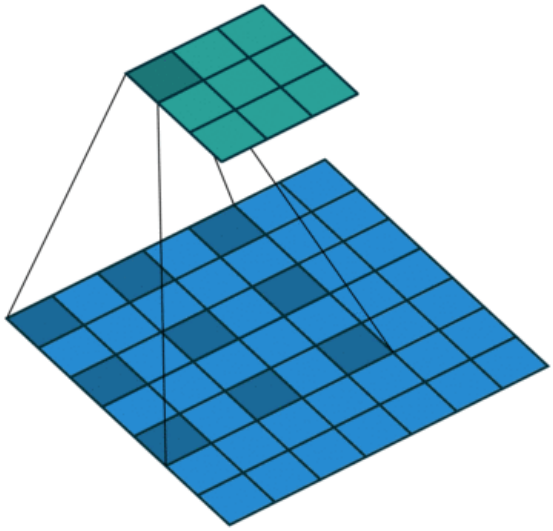
kde n, $m \subset \mathbb{Z}$. Konvolúciu môžme interpretovať ako posúvanie otočeného lineárneho filtra cez vektor. Interpretáciu je možné jednoducho rozšíriť do 2D, posúvaním otočeného 2D lineárneho vektora cez maticu. V konvolučných vrstvách neurónových sietí zanedbávame otáčanie filtra pretože parametre filtra sa upravujú samostatne pomocou optimalizéra s využitím metód „back propagation“. Na základe tejto definície výsledkom konvolúcie je matica, ktorá je na každej strane rozšírená o polovicu veľkosti filtra. Výstupom konvolúcie 3×3 je teda matica $(3 + 2) \times (3 + 2)$, po zaokrúhlení. Pri konvolúcii sa používa niekoľko dôležitých parameterov. Jeden z nich je „stride“. Tento parameter definuje o koľko sa posunie konvolučný filter po vstupnom vektore. Pri hodnote 1 pre parameter „stride“ konvolúcia zachová pomer strán vstupu (ako vstup predpokladáme obrázok - 2D vektor), mení sa iba jeho hĺbka podľa množstva použitých filtrov. Ak uvažujeme s hodnotou inou ako 1 veľkosť vstupu sa zmení nasledovne $(h/stride, w/stride)$, kde h a w je výška a šírka vstupu. V nasledujúcom texte, ak nie je uvedené inak, uvažujeme s hodnotou 1 pre tento parameter. Ďalším z parametrov je „padding“. Môže nadobúdať hodnoty „SAME“ alebo „VALID“. Podrobný popis možností tohto parametra a jeho vplyv na výstup je popísaný v [6]. V práci bola v každej konvolučnej vrstve použitá hodnota „SAME“ pre parameter „padding“. Pri návrhu konvolučných sietí a zvlášť pri sieťach, ktoré sú určené na detekciu objektov je dôležité, aby každý neurón vo výstupnej vrstve reprezentoval určitú oblasť vstupného vektora. Táto vlastnosť sa nazýva zorné pole (FOV). Zorné pole neurónov je možné rozšíriť zväčšením konvolučného filtra alebo použitím niekoľkých konvolučných vrstiev za sebou. Príklad takto použitých konvolučných vrstiev je na obrázku 15 [2]



Obr. 15: Zorné pole (FOV) konvolučných vrstiev. Obrázok zobrazuje veľkosť FOV, ktorú je možné dosiahnuť použitím iba troch konvolučných vrstiev. Jednotlivé vrstvy využívajú konvolúciu 3×3 s parametrom $stride = 2$. Výstupná vrstva (oranžová), ktorá je pomerne malá (iba 2×2) nesie informácie z prvkov vstupného vektora (šedá, 7×7)

[2] https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Zorné pole neurónov v konvolučných vrstvách je možne ešte zväčšiť použitím rozšírenej konvolúcie („dilated convolution“) alebo „atrous“ konvolúcie [27]. Rozšírená konvolúcia využíva medzery medzi jednotlivými prvkami filtra. Rozšírená konvolúcia teda pokryje oveľa väčšiu časť vstupu ako obyčajná konvolúcia. Použitie viacerých rozšírených konvolúcii za sebou zväčší zorné pole neurónov oveľa rýchlejšie ako v prípade zreťazenia obyčajných konvolučných vrstiev. Môže sa zdať, že rozšírená konvolúcia stratí podstatnú časť údajov a tiež ovplyvní veľkosť výstupného vektora. Využitie parametrov „stride“ a „padding“ zabráni strate údajov a tiež zabezpečí, že veľkosť výstupu ostane nezmenená. Príklad rozšírenej konvolúcie je zobrazený na obrázku 16[3]



Obr. 16: Zorné pole (FOV) rozšírenej konvolučnej vrstvy. Obrázok zobrazuje veľkosť FOV, ktorú je možné dosiahnuť použitím rozšírenej konvolučnej vrstvy. Vrstva využíva konvolúciu 3×3 s rozšírením $dilation = 2$

Pri návrhu neurónových sietí je veľmi dôležité vedieť, čo má daná sieť detegovať a ako bude vyzerať jej výstup. Konvolučné siete, ktoré detegujú objekty rôznych veľkostí sú založené práve na zornom poli (FOV) jednotlivých neurónov výstupnej vrstvy. Preto je dôležité vedieť ako vypočítať (FOV) pre tieto vrstvy aby detegovali objekty určitej veľkosti. Na výstupe konvolučnej siete je hodnota, ktorá reprezentuje časť vstupného vektora s veľkosťou $s \times s$. „down-sampling“ siete s je daný počtom „pooling“ vrstiev a parametrom konvolúcie „stride“. Zorné pole siete je možné definovať ako časť vstupného vektora, ktorá ovplyvňuje jeden prvok vo výstupnej vrstve. Ako bolo spomenuté, zreťazenie konvolúcii zvýši zorné pole siete. Jedna konvolučná vrstva môže mať zorné pole napríklad 3×3, zatiaľ čo dve zreťazené konvolučné vrstvy majú zorné pole 5×5. Je to spôsobené tým, že každý prvok na vstupe druhej vrstvy už je ovplyvnený susednými prvkami z prvej vrstvy. Je teda veľmi dôležité poznať zorné pole každej konvolučnej vrstvy, pretože to určuje aké objekty budú danou vrstvou detegované. Navrhnutá sieť je plne

[3] https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

konvolučná, teda zorné pole sa s pribúdajúcimi vrstvami zvyšuje. Sieť deteguje objekty rôznej veľkosti a preto musia byť výstupné vrstvy dobre umiestnené vrámci architektúry siete. Aby bolo možné vypočítať zorné pole jednotlivých vrstiev je dôležité poznať parameter s_i „down-sampling“, veľkosť filtra konvolúcie k_i a parameter rozšírenej konvolúcie d_i. Ak parameter $d_i = 1$ jedná sa o obyčajnú konvolúciu. Spojením týchto parametrov dostaneme rovnicu

$$fov_i = s_i(d_i(k_i - 1) + 1) - s_{i-1} + fov_{i-1}\,,$$

kde i označuje číslo vrstvy. „pooling“ vrstva alebo zvýšenie parametru „stride“ je reprezentované zvýšením premennej s_i. Využitím vyššie uvedenej rovnice a dosadením parametrov je možné vypočítať zorné pole (FOV) pre každú vrstvu siete

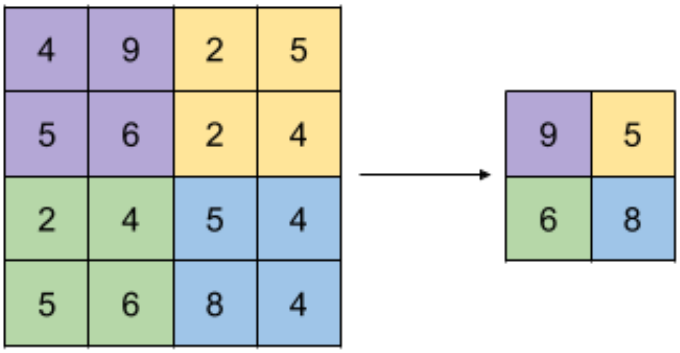
$$\begin{aligned}
fov_0 &= 0, s_0 = 0\,,\\
fov_1 &= 1(1(3-1)+1) - 0 + 0 = 3\,,\\
fov_2 &= 1(3(3-1)+1) - 1 + 3 = 9\,,\\
fov_3 &= 2(1(3-1)+1) - 1 + 9 = 14\,,\\
fov_4 &= 2(2(3-1)+1) - 2 + 14 = 22\,,\\
fov_5 &= 2(4(3-1)+1) - 2 + 22 = 38\,,\\
fov_6 &= 2(7(3-1)+1) - 2 + 38 = 66\,,\\
fov_{output_2} &= 2(1(1-1)+1) - 2 + 66 = 66\,,\\
fov_7 &= 4(1(3-1)+1) - 2 + 66 = 76\,.
\end{aligned}$$

Za každou konvolučnou vrstvou, okrem výstupnej, bola použitá aktivačná funkcia. V Tomto prípade bola použitá $ReLU$ aktivačná funkcia pretože sa ukázala ako najvhodnejšia voľba pri spracovaní obrazu.

4.4.3 „Pooling“ vrstva

„Pooling“ vrstva sa vo všeobecnosti využíva na redukciu dimenzii (veľkosti) vstupného vektora. Existuje niekoľko druhov takýchto vrstiev, napríklad „AveragePooling2D“ alebo „MaxPool2D“. Najčastejšie používaný je práve „MaxPool2D“ a preto bol použitý aj v tejto práci. Vrstva bola použitá medzi časťami siete, ktoré definovali jednotlivé mierky. Bol použitý filter s veľkosťou 2×2 a parametrami $stride = 2$ a $padding = SAME$. Parametre „stride“ a „padding“ majú rovnakú funkciu ako pri konvolučných vrstvách. Príklad „MaxPool2D“ vrstvy je zobrazený na obrázku 17[4].

[4]https://towardsdatascience.com/deep-learning-personal-notes-part-1-lesson-3-cnn-theory-convolutional-filters-max-pooling-dbe68114848e



Obr. 17: Príklad „MaxPool2D“ vrstvy, s parametrom $stride = 2$ a $padding = SAME$, ktorej výstupom je matica s polovičnou veľkosťou. Zo vstupu je vybraný najvyšší prvok vrámci každého posunutia filtra a ten je uložený ako výstup.

4.4.4 Výstupná vrstva

Ako už bolo spomenuté, reprezentácia ohraničujúcich kvádrov vo výstupnej vrstve je odvodená od použitej metódy DenseBox [13]. Výstup obsahuje 8 vrstiev, kde prvá z nich je pravdepodobnostná vrstva a ostatné vrstvy obsahujú informácie o súradniciach bodov. Príklad takto upraveného výstupu je možné vidieť na obrázku 3. Veľkosť výstupu je určená mierkou s. DenseBox metóda využíva mierku 4, keďže ale chceme pri jednom prechode sieťou detegovať objekty rôznych veľkostí, boli použité mierky (2, 4, 8, 16). Informácie pre každú mierku boli extrahované z rôznych častí siete ako je to naznačené na obrázku 13.

4.5 Implementácia modelu

Model bol implementovaný podobne ako stratová funkcia 4.3. Verzia frameworku TF 1.x nepodporovala OOP prístup a tak jednotlivé vrstvy boli zreťazené za sebou tak, aby vytvorili požadovanú architektúru. Nová verzia frameworku podporuje OOP prístup pri vytváraní modelov ale aj jednotlivých vrstiev. Pre všetky vrstvy boli vytvorené „callable“ objekty, ktoré sú potom použité v navrhnutom modeli. Takýto objekt bol vytvorený aj pre samotný model. Model zapuzdruje použité vrstvy a celý proces spracovania vstupných dát. TF 2.x nevyžaduje popísaný postup, je možné definovať vrstvy samostatne a zreťaziť ich ako v prípade staršej verzie. V tomto prípade je nutné si dať pozor na premenné, ktoré budú upravované počas „back-propagation“. S využitím OOP sa o tieto premenné postará abstraktná trieda, z ktorej je model odvodený. Využitie OOP prístupu sa ukázalo ako veľmi jednoduché a prehľadné riešenie, preto bol tento postup použitý pri implementácii.

5 Spracovanie výstupu

Po úspešnom trénovaní siete, je nutné zistiť presnosť siete, odstrániť nevyhovujúce výsledky a zobraziť ohraničujúce kvádre, ktoré majú najvyššiu pravdepodobnosť v prvej vrstve výstupu. Z výstupu boli vybrané pixely, ktorých pravdepodobnosť je viac ako zvolená pravdepodobnosť. Hodnota pravdepodobnosti môže byť ľubovoľná. Je však dôležité si uvedomiť, že ak hodnota pravdepodobnosti bude príliš vysoká, môžu byť z výstupu odstránené objekty, ktorých pravdepodobnosť nie je tak vysoká, napriek tomu že objekt je detegovaný správne. Takéto správanie môže byť spôsobené napríklad prekrytím objektu alebo inými vonkajšími vplyvmi, ktoré by mohli ovplyvniť viditeľnosť objektu. V druhom prípade hodnota pravdepodobnosti nemôže byť ani príliš malá, pretože celkový počet nájdených ohraničujúcich kvádrov by bol príliš veľký. V tomto prípade by výstup mohol obsahovať aj nesprávne identifikované objekty. Po zvážení všetkých faktorov, ktoré môžu ovplyvniť výstup bola zvolená hodnota pravdepodobnosti 70%.Výsledkom je výstup, ktorý obsahuje iba pixely, ktorých hodnoty sú vyššie ako nejaká prahová hodnota. Prahovú hodnotu (treshold) je možné vypočítať nasledovne

$$treshold = \frac{max(out^0) * prob\_value}{100},$$

kde $max(out^0)$ je maximálna hodnota z prvej vrstvy výstupu, $prob\_value$ je zvolená hodnota pravdepodobnosti. $treshold$ predstavuje prahovú hodnotu, podľa ktorej boli pixely s hodnotou menšou ako je táto prahová hodnota odstránené z výstupu tak, že hodnoty pixelov vo všetkých vrstvách na daných súradniciach boli nastavené na hodnotu 0 aby neovplyvnili ďalšie spracovanie. Ďalším krokom spracovania výstupu je získanie pôvodných hodnôt súradníc. Počas trénovania boli súradnice normalizované do rozsahu $\langle 0, 1 \rangle$ podľa kapitoly 4.2.1. Na získanie pôvodných hodnôt súradníc bol použitý opačný proces ako pri normalizácii. V priebehu de-normalizácie sú objekty zhlukované do skupín podľa najbližšieho lokálneho minima v pravdepodobnostnej vrstve. Informácia do ktorej skupiny objekt patrí ďalej slúži na výber najlepšieho z nich a na výpočet presnosti vzhľadom k ohraničujúcemu kvádru z KITTI datasetu podľa kapitoly 5.1.V priebehu vytvárania programu a testovania sa na výstupe často objavovali ohraničujúce kvádre, ktoré boli pre danú mierku príliš male (veľké) alebo tvarom neodpovedali hľadaným objektom. Z KITTI datasetu bola preto extrahovaná minimálna a maximálna výška objektov vzhľadom na mierku. Príklad minimálnej a maximálnej výšky objektov pre mierku 2

- Vstup: O objekty pre mierku 2
- Inicializácia: H = []
- For objekt in O
    1. $H <= abs(objekt.fbl_y - objekt.ftl_y)$

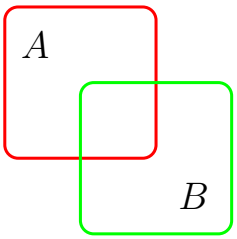
- $min = avg(H) - 2 * E(H)$
- $max = avg(H) + 2 * E(H)$
- return min, max

Ak je výška objektu vo výstupe mimo tento rozsah, objekt je odstránený. Po tomto kroku je možné vykonať inverznú projekciu na zvyšných objektoch 3.5.

5.1 Presnosť siete

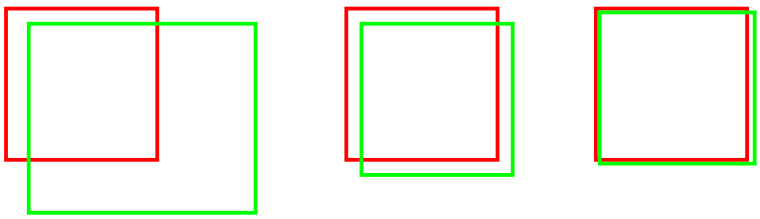
Pri detekcii 2D objektov sa najčastejšie používa algoritmus IoU - „Intersection over Union“. IoU prístup dáva do pomeru oblasť, ktorú majú ohraničujúce štvoruholníky spoločnú a oblasť, ktorú zaberajú spoločne. V prípade 2D majú ohraničenia iba 4 stupne voľnosti (pozícia, veľkosť) a výpočet jednotlivých oblastí je triviálny.

$$\text{IoU} = \frac{A \cap B}{A \cup B}$$



Obr. 18: Všeobecný postup výpočtu IoU. Zdroj [21]

IoU je možné chápať ako zhodu predikovaného objektu s objektom z anotácii datasetu. Príklad IoU pre 2D objekty je na obrázku 19.



IoU: 0, 4034 **Slabá**

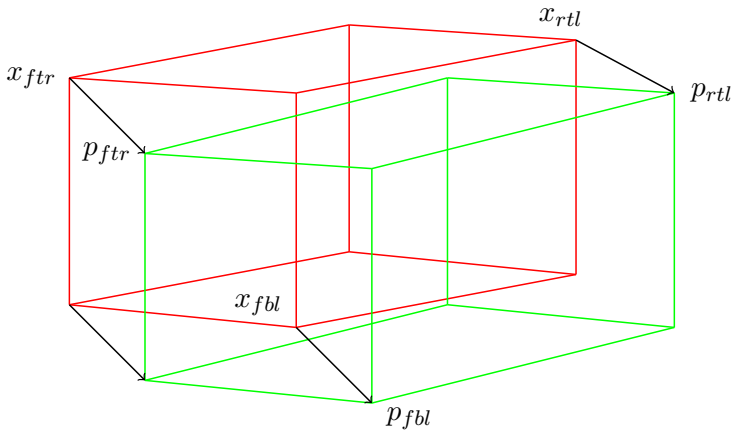
IoU: 0, 7330 **Dobrá**

IoU: 0, 9264 **Výborná**

Obr. 19: Príklad IoU pre rôzne ohraničenia. Zdroj [21].

Práca je však zameraná na detekciu 3D ohraničujúcich kvádrov, ktoré majú v tomto prípade 7 stupňov voľnosti (1). Z tohto dôvodu nebolo možné použiť IoU postup ako pri 2D objektoch. Ohraničujúce kvádre, ktoré sa snažíme predikovať sú orientované. Tzn. majú oddelenú prednú a zadnú stenu kvádra. Ak by bolo možné upraviť IoU tak aby dokázal pracovať so 7 stupňami voľnosti nastal by problém s orientáciou kvádra. V prípade rovnakej pozície a veľkosti predikovaného kvádra a kvádra z anotácii datasetu a opačnej orientácii predikovaného kvádra, výsledok

IoU by bol rovnaký ako keby boli oba kvádre orientované správne. Bol navrhnutý nový prístup ako zistiť zhodu medzi dvomi 3D ohraničujúcimi kvádrami PDE (point distance evaluation), založený na vzdialenosti dvoch bodov. Na vstupe algoritmu sú dva ohraničujúce kvádre, jeden z výstupu siete, druhý z anotácii datasetu. Každý z nich obsahuje súradnice všetkých vrcholových bodov.



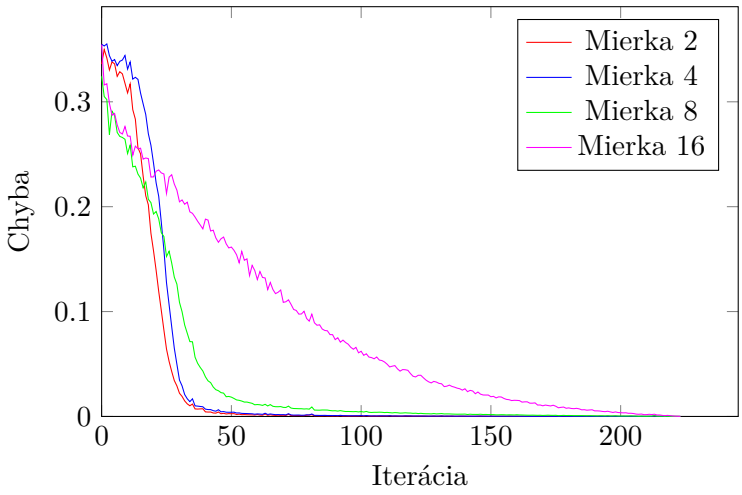
Obr. 20: Príklad PDE na dvoch ohraničujúcich kvádroch s čiastočným prekrytím.

Jednotlivé vrcholové body sa spárujú podľa ich pozície na ohraničujúcom kvádri ako je to naznačené na obrázku 20. Vrchol X_{ftr} bude spárovaný s vrcholom P_{ftr}. Takto navrhnuté párovanie bodov odstráni problem s orientáciou kvádra. V prípade, že by kvádre mali opačnú orientáciu výsledná vzdialenosť by bola oveľa vyššia ako v prípade ak by boli orientované rovnakým smerom. Výsledkom je pomer vzdialeností jednotlivých bodov a počtu objektov na obrázku. Výsledná vzdialenosť je z intervalu $\langle 0, \inf \rangle$. 0 predstavuje úplné prekrytie ohraničujúcich kvádrov (100 % úspešnosť), v opačnom prípade čím vyššia vzdialenosť tým vyššia chyba siete.

5.2 Zhodnotenie výsledkov

Sieť bola z dôvodu nedostupnosti výkonnejšieho zariadenia trénovaná len na malom datasete. Dataset obsahoval 100 trénovacích dát, ktoré boli počas trénovania náhodne vyberané a spracovávané. Sieť je rozdelená na 4 časti podľa definovaných mierok 5. Každá časť má vlastný optimalizér a chybovú funkciu. Priebeh znižovania chybovosti siete je zobrazený na obrázku 21. Sieť už takmer na začiatku dosiahla veľmi nízku chybovosť, avšak výsledky v prvých častiach siete sú stále dosť zašumené 22. Môže to byť spôsobené práve malým datasetom. V tretej a štvrtej časti sú však výsledky dostatočne presné. Na optimalizáciu bol použitý SGD (Stochastic Gradient Descent) optimalizér, ktorý bol omnoho stabilnejší počas trénovania ako Adam optimalizér. Pri dlhšom trénovaní a rozsiahlejšom datasete by sieť mala byť schopná naučiť sa detegovať objekty v neznámych dátach. Sieť dokáže detegovať objekty rýchlo a dostatočne presne čo ukázal test pri trénovaní na jednom obrázku. Čas detekcie všetkých objektov v jednom obrázku je približne 2 s. Trénovanie a testovanie siete prebiehalo na grafickej karte *NVIDIA GEForce GTX 970m*,

ktorá je už pomerne stará a oproti nový a výkonnejším grafickým kartám značne pomalšia. S výkonnejším HW by testovanie nemuselo byť dlhšie ako 1 s.



Obr. 21: Chyba siete počas trénovania na malom datasete. Hodnoty sú normalizované do intervalu $\langle 0, 1 \rangle$ pre lepšie znázornenie

(a) Mierka 2 (b) Mierka 4

(c) Mierka 8 (d) Mierka 16

Obr. 22: Výsledok siete po trénovaní na malom datasete. Po odstránení nevyhovujúcich objektov z výstupu, je výstup v prvých častiach siete stále dosť zašumený, čo je spôsobené práve malým datasetom

Implementácia v oboch verziách frameworku a návod na použitie sú súčasťou práce ako prílohy. Prílohy sú taktiež verejne dostupné na stránke [5].

[5] https://github.com/mrvecka/Object-Detection

6 Experimenty

V nasledujúcej kapitole sú popísané možnosti úpravy implementácie siete prípadne úpravy samotnej architektúry siete.

6.1 Zrýchlenie siete

TensorFlow framework priniesol s verziou 2.x veľa nových funkcii a možností. V prípade TF 1.x sú všetky premenné používané počas trénovania globálne. V TF 2.x premenné majú svoj „scope“ vrámci funkcii, kde ich je možné používať. Nová verzia frameworku ponúka tiež jednoduché možnosti krokovania („debug“ režim). Všetky tieto možnosti a funkcie súvisia s príchodom tzv. „eager execution“. Avšak najzaujímavejšou funkciou je $tf.function$. Framework si na pozadí vytvára graf operácii, ktoré následne vykonáva. Takýto graf operácii je prehľadný a je jasne daná následnosť operácii aj v prípade vetvenia programu. Pre funkcie takto označené je vytvorený graf operácii, ktorý je podgrafom celého vykonávacieho cyklu. Príklad takto označenej funkcie je uvedený vo výpise 5.

```
@tf.function
def call(target, output):
```

Výpis 5: Príklad použitia $tf.function$

Napriek všetkým výhodám, ktoré $tf.function$ ponúka nemá zmysel ju používať na každú funkciu. Takto označovať by sa mali len funkcie v ktorých dochádza k náročným operáciám. Je to z toho dôvodu, že niektoré funkcie nie sú vôbec časovo náročné a takéto označenie by nemalo žiaden význam, navyše časové zlepšenie po prevode by bolo zanedbateľné. Funkcie, ktorých parametre sa menia počas behu programu by tiež nemali byť takto označované. V tomto prípade by sa vytváranie grafu pre funkciu vykonávalo vždy po zmene vstupných parametrov čo spomalí jej vykonávanie. Niekedy je ale výhodné mať takto označenú funkciu a zároveň parametre, ktoré sa menia počas behu programu. V tomto prípade je možné využiť špecifickú konštrukciu tohto označenia a využiť parameter „input_signature“[6], ktorým je označený vstupný meniaci sa parameter čo zabráni vytváraniu grafu pri každom spustení. Porovnanie modelov počas trénovania je zobrazené v tabuľke 6. Tabuľka zobrazuje časový rozdiel frameworkov ale aj modelov bez/s funkciami $tf.function$. Môžeme si všimnúť, že novšia verzia frameworku bez dodatočnej úpravy funkcii je oveľa rýchlejší ako jeho staršia verzia (stĺpec 3). Avšak nová verzia frameworku s upravenými funkciami je skoro dvakrát rýchlejšia (stĺpec 4). V tomto prípade prvých 50 iterácii bolo vykonaných s časom 48 s, čo je najviac v porovnaní s ostatnými. Dôvod tak vysokého času je práve vytváranie grafov pre jednotlivé funkcie počas prvého prechodu. Dlhší čas inicializácie a prípravy modelu je však zanedbateľný v porovnaní s takmer dvoj násobným zrýchlením počas každej ďalšej iterácie.

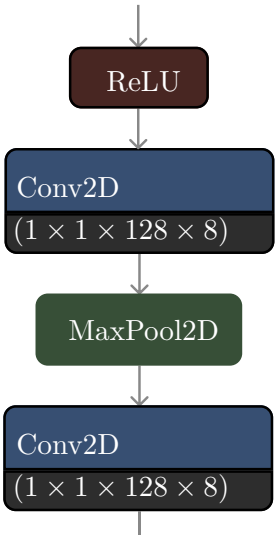
[6] https://www.tensorflow.org/api_docs/python/tf/function

Iterácia	TF 1.x (s)	TF 2.x (s)	TF 2.x (s)*
50	37	13	48
100	40	11	6
150	38	11	6
200	38	11	6

Tabuľka 6: Porovnanie časovej náročnosti modelov s využitím funkcii TF 2.x. Posledný stĺpec definuje model, ktorý využíva funkcie označené $tf.function$. Časové údaje sú zaokrúhlené na sekundy. Pri testovaní modelov bol použitý rovnaký obrázok ($batch\_size = 1$) a rovnaká architektúra siete

6.2 Vplyv výstupnej vrstvy na ďalšie spracovanie

Architektúra siete je inšpirovaná prácou [18]. Jej popis a grafické znázornenie je zobrazené v kapitole 4.4. Zo siete sú v priebehu prechodu vstupných dát v určitých miestach (obrázok 13) extrahované údaje, z ktorých sú následne získavané ohraničujúce kvádre. V týchto miestach sa sieť rozdeľuje ako je to naznačené na obrázku 14. Takéto rozdelenie vytvorí jednu vrstvu, ktorá nemá vplyv na ďalšie spracovanie a spomaľuje sieť. V tejto kapitole som sa zameral na odstránenie tohto rozdelenia. Upravená časť siete je zobrazená na obrázku 23. Výstupná vrstva nemohla byť úplne odstránená, pretože jej výstup je stále potrebný pre získanie ohraničujúcich kvádrov. Výstupná vrstva po úprave ovplyvňuje ďalšie spracovanie vstupných dát. Po trénovaní a porovnaní výsledkov, upravená sieť dosahovala v niektorých prípadoch o niečo lepšie výsledky. Predpokladám, že to je spôsobené referenčnými dátami. Objekty boli vkladané do referenčných dát s určitým prekrytím, teda niektoré objekty sa môžu nachádzať v dvoch susedných mierkach naraz. Tento fakt spôsobí, že ak prvá časť siete tento objekt už detegovala, nasledujúca časť siete, ktorá podľa referenčných dát tiež obsahuje rovnaký objekt, nemusí tento objekt detegovať znova. Nasledujúca časť siete len upraví oblasť, kde sa objekt nachádza, pretože pre túto časť môžme zvoliť odlišný okruh od stredu objektu 4.2. Zlepšenie výsledkov môže byť spôsobné aj pridanou konvolučnou vrstvou. Nejedná sa však o celkom novú vrstvu, ale o výstupnú vrstvu, ktorá má teraz vplyv na ďalšie spracovanie. Každá konvolučná vrstva obsahuje trénovacie parametre, ktoré sa upravujú počas „back propagation“. Tieto parametre v prvom návrhu siete nemali vplyv na ďalšie spracovanie, v upravenej architektúre však ponúkajú priestor na zlepšenie pre sieť. Počas trénovania a testovania vplyvu tejto úpravy na výsledok som si všimol aj mierne zlepšenie v rýchlosti siete. Toto zlepšenie predstavuje asi $0.5 - 1\,s$ (3. stĺpec, tabuľka 6 obsahuje výsledky pre sieť pred úpravou) pre jeden obrázok. V prípade zvýšenia rozsahu vstupných dát ($batch\_size > 1$) by toto zrýchlenie bolo viditeľnejšie.



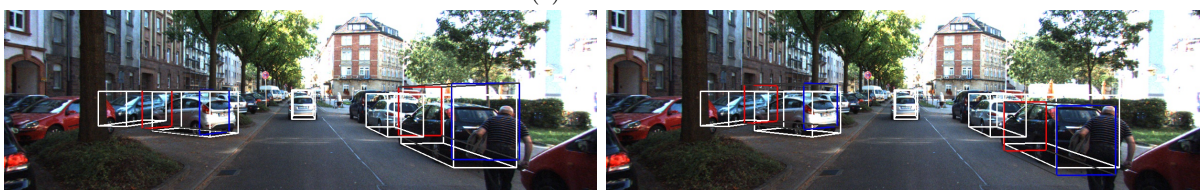
Obr. 23: Odstránenie rozdelenia siete pri získavaní výstupu. „ReLU“ predstavuje výstup predchádzajúcej vrstvy, v tomto prípade vrstvy 6. Nasledujúca vrstva „Conv2D“ je výstupná vrstva pre mierku 2. Výstup tejto vrstvy je použitý ako vstup nasledujúcej „MaxPool“ vrstvy, ktorá pokračuje v spracovaní pre ďalšie mierky

Výsledok upravenej siete na testovacom obrázku 000008 je zobrazený na obrázku 24. Upravená verzia siete dosahovala v niektorých prípadoch lepšie výsledky, no v niektorých sa zhoršila. Nedá sa teda všeobecne povedať, že by takáto úprava výrazne pomohla pri detekcii objektov. Ako už bolo spomenuté niektoré objekty sa môžu nachádzať zároveň v dvoch susedných mierkach. Tento fakt je výhodou aj nevýhodou navrhnutej úpravy. Zatiaľ čo v prípade objektov, ktoré sa nachádzajú v oboch susedných mierkach dochádza k zlepšeniu, v prípade objektov, ktoré sa nachádzajú len v jednej dochádza k zhoršeniu. Môžme povedať, že v prvom prípade referenčné dáta spolu súvisia, pretože sa snažia detegovať rovnaké objekty a sieť v nasledujúcej časti len mierne upraví výstup. V druhom prípade referenčné dáta spolu nesúvisia a sieť musí detegovať nové objekty. Ako príklad môžme uviesť obrázok, na ktorom budú dve vozidlá. Prvé vozidlo sa bude nachádzať v ľavej časti a patrí do mierky 2, druhé vozidlo sa nachádza v pravej časti a patri do mierky 4. Sieť sa musí naučiť detegovať tieto odlišnosti a to vedie k zvýšenej chybe. Vzhľadom na dosiahnuté výsledky a fakt, že počet objektov, ktoré sa nachádzajú len v jednej mierke je oveľa väčší ako počet objektov, ktoré sa nachádzajú v dvoch, bola táto úprava odstránená z konečného návrhu siete.

(a) Mierka 2

(b) Mierka 4

(c) Mierka 8

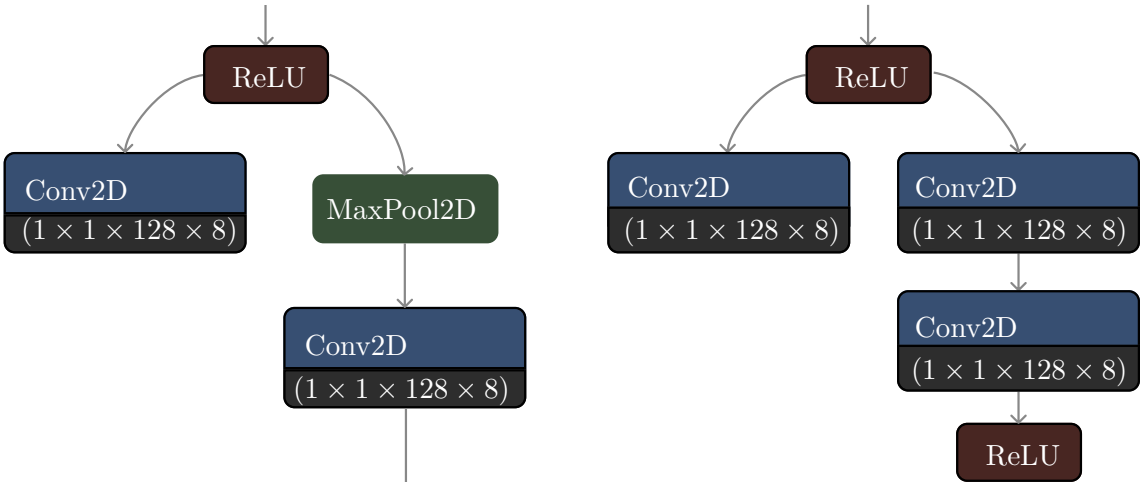


(d) Mierka 16

Obr. 24: Porovnanie výsledkov po odstránení rozdelenia siete pre jednotlivé mierky. Vľavo výsledok siete pred rozdelením, vpravo výsledok siete po rozdelení. Ohraničenia bielou farbou znázorňujú ohraničenia z datasetu

6.3 MaxPool vs konvolúcia

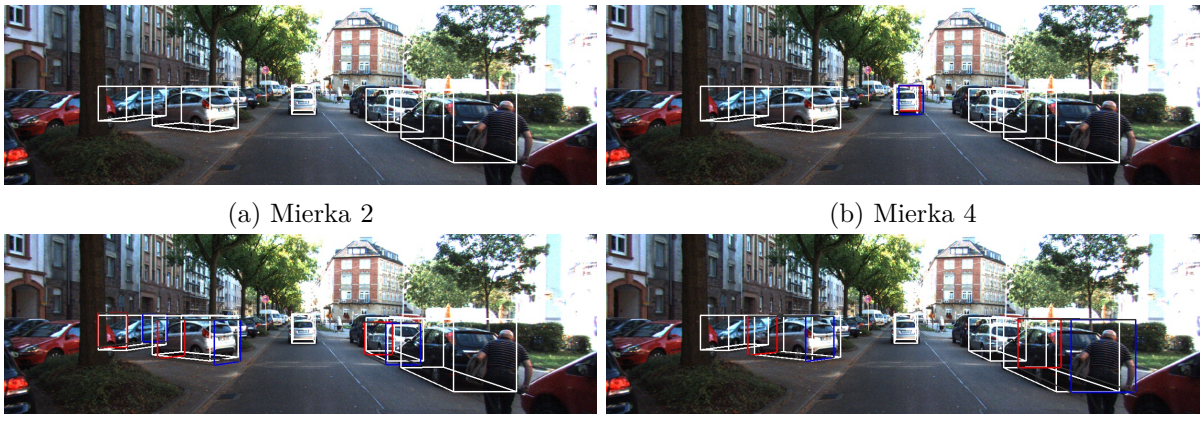
Navrhnutá sieť deteguje objekty rôznych veľkostí. Aby sme dokázali detegovať objekty v jednom prechode, musela byť sieť rozdelená na niekoľko častí, Výstupy z týchto častí obsahujú objekty, ktoré do nej podľa svojej veľkosti patria. Prechod medzi jednotlivými časťami je zabezpečený vrstvou „MaxPool2D“. Vrstva zmenší výšku a šírku vstupných dát na polovicu, hĺbka je zachovaná. Princíp a fungovanie vrstvy je popísaný v kapitole 4.4.3. Vrstva neobsahuje žiadne trénovacie parametre a len slepo spracuje vstup. V priebehu „back propagation“ nie sú v tejto vrstve žiadne parametre, ktoré by boli upravené oproti konvolúcii, ktorá tieto parametre obsahuje. V nasledujúcej kapitole som sa zaoberal rozdielom medzi použitím „MaxPool2D“ vrstvy a „Conv2D“ vrstvy. Efekt „MaxPool2D“ vrstvy sa dá dosiahnuť použitím konvolúcie s parametrom $stride = 2$. Navyše konvolúcia obsahuje trénovacie parametre, ktoré by mohli zvýšiť presnosť siete. Podľa [25] výmena vrstiev „MaxPool2D“ za konvolúciu zvýšila presnosť testovacích sietí, preto som sa rozhodol upraviť sieť podľa tejto teórie. Upravená časť siete je na obrázku



(a) Pred úpravou (b) Po úprave

Obr. 25: Odstránenie „MaxPool2D“ vrstiev zo siete. Na obrázku je zobrazený prechod medzi mierkami 2 a 4. Na obrázku (b) bola vrstva „MaxPool2D“ nahradená trojicou „Conv2D“, „BiasAdd“ a aktivačnou funkciou „Relu“. Po tejto trojici pokračuje spracovanie v ďalšej časti, ktorá ostala nezmenená

Výsledky po opätovnom trénovaní a testovaní siete sú zobrazené na obrázku 26. V porovnaní s obrázkom 24 (vpravo) sú výsledky viditeľne o niečo presnejšie. Po úprave sa čas trénovania mierne zvýšil, pretože operácia konvolúcie je náročnejšia ako operácia „MaxPool2D“. Dôležite je si uvedomiť, že po pridaní konvolučnej vrstvy sa zmení aj FOV siete 4.4.2. Keďže navrhnutá úprava priniesla zlepšenie výsledkov a len minimálne zvýšenie času detekcie, architektúra 4.4 bola upravená a konečný návrh siete obsahuje spomínanú úpravu.



(a) Mierka 2 (b) Mierka 4

(c) Mierka 8 (d) Mierka 16

Obr. 26: Výsledok siete po odstránení vrstvy „MaxPool2D“ pre všetky mierky

7 Záver

V práci sme sa zaoberali detekciou 3D objektov v obraze. Snažili sme sa pomocou navrhnutej siete detegovať vozidlá v obraze a okolo nájdených objektov vykresliť ohraničujúce kvádre. Z uskutočneného prieskumu sme vybrali vhodnú metódu na detekciu, ktorú sme spojili s neurónovou sieťou a tým sme detegovali objekty. Informácie z KITTI datasetu sme upravili do podoby vhodnej na spracovanie touto metódou. Odstránili sme z anotácii objekty, ktoré sa priveľa prekrývali alebo boli v obraze len čiastočne. Využili sme nový formát súboru, v ktorom boli iba dôležité informácie potrebné na spätné zrekonštruovanie ohraničujúcich kvádrov. Je možné použiť akýkoľvek dataset, z ktorého vieme získať údaje v podobe ako popisuje dokument v kapitole 3.4. Môžme si vytvoriť aj vlastný dataset, ktorý bude obsahovať len tie údaje, ktoré potrebujeme. Z očakávaných výsledkov neurónovej siete sme dokázali rekonštruovať celý ohraničujúci kváder podľa postupu v kapitole 3. Získané ohraničenia sme filtrovali podľa pravdepodobnosti s akou obsahujú objekty a veľkosti ohraničení podľa mierky, v ktorej sa nachádzali. Výsledkom je program, ktorý upraví informácie z datasetu a spustí trénovanie navrhnutej siete. Sieť bola v prvom kroku navrhnutá v staršej verzii frameworku. S drobnými úpravami architektúry a implementácie bola sieť prevedená do novšej verzie, čo prinieslo značné zrýchlenie. Výsledok siete po trénovaní a testovaní je popísaný v kapitole 5.2. Sieť je dostatočne rýchla a presná. Bohužiaľ z dôvodu nedostupnosti výkonnejšieho počítača, bola sieť trénovaná len na malom datasete. Aktuálny model, trénovaný na malom datasete nedosahuje taký stupeň všeobecnosti, aby sa dal použiť na neznáme dáta. Toto trénovanie však ukázalo, že sieť funguje a je schopná sa naučiť detegovať objekty. Spustenie trénovania na rozsiahlejšom datasete by teda malo priniesť model, ktorý dokáže detegovať objekty v neznámych vstupných dátach. Rýchlosť detekcie by bolo možné vylepšiť použitím distribuovanej stratégie. Sieť je rozdelená na 4 časti, z ktorých každá by dokázala fungovať samostatne s využitím nejakej vstupnej fronty. Takto by prvá časť siete po ukončení svojej úlohy nemusela čakať kým ostatné časti spracujú dáta ale mohla by začať spracovávať ďalšiu skupinu dát. Podobný princíp by sa dal využiť aj pri testovaní, čo by prinieslo zvýšenie FPS v prípade spracovania videa.

Literatúra

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[3] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016.

[4] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015.

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. pages 1907–1915, 2017.

[6] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv 1603 07285*, 2016.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[9] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9), 2015.

[13] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection, 2015.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[15] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 1989.

[16] Robert C. Bolles Martin A. Fischler. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, pages 381–395, 1981.

[17] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017.

[18] Libor Novak. *Vehicle detection and pose estimation for autonomous driving*. PhD thesis, PhD thesis, Masters thesis, Czech Technical University in Prague, 2017 . . . , 2017.

[19] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[21] Adrian Rosebrock. Intersection over union (iou) for object detection. `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`, 2016.

[22] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR 2011*. IEEE, 2011.

[23] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv 1409 1556*, 2014.

[25] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[26] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2), 2013.

[27] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv 1511 07122*, 2015.